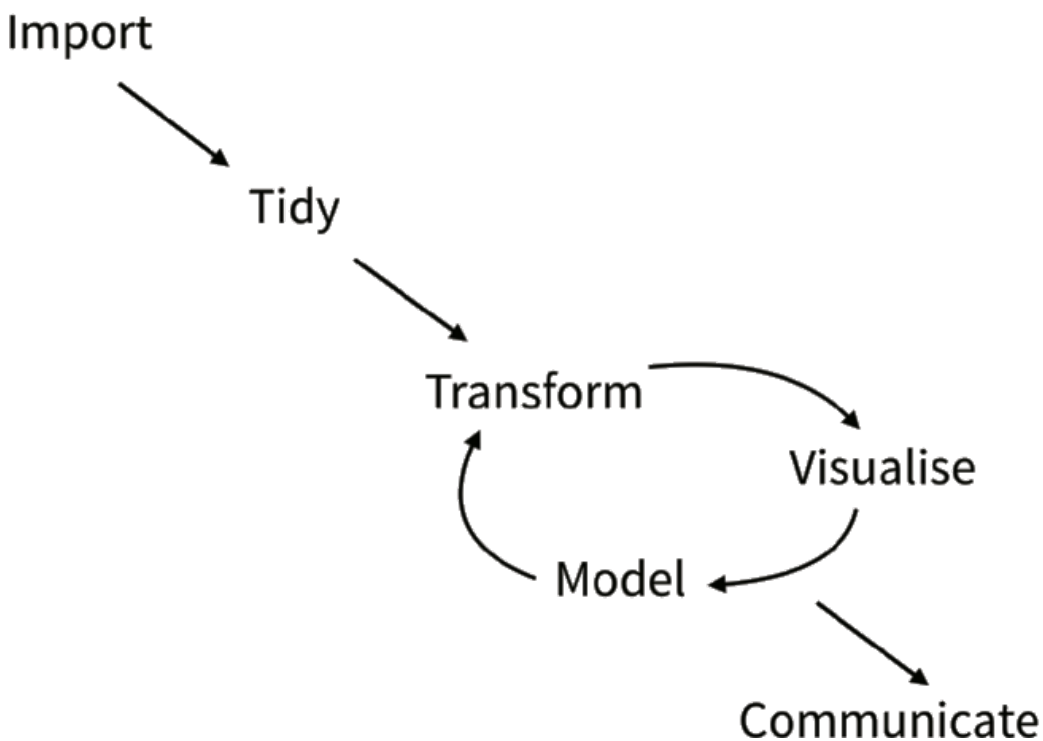
Manage your analyses workflows with the drake R package

Grenoble R Users Group

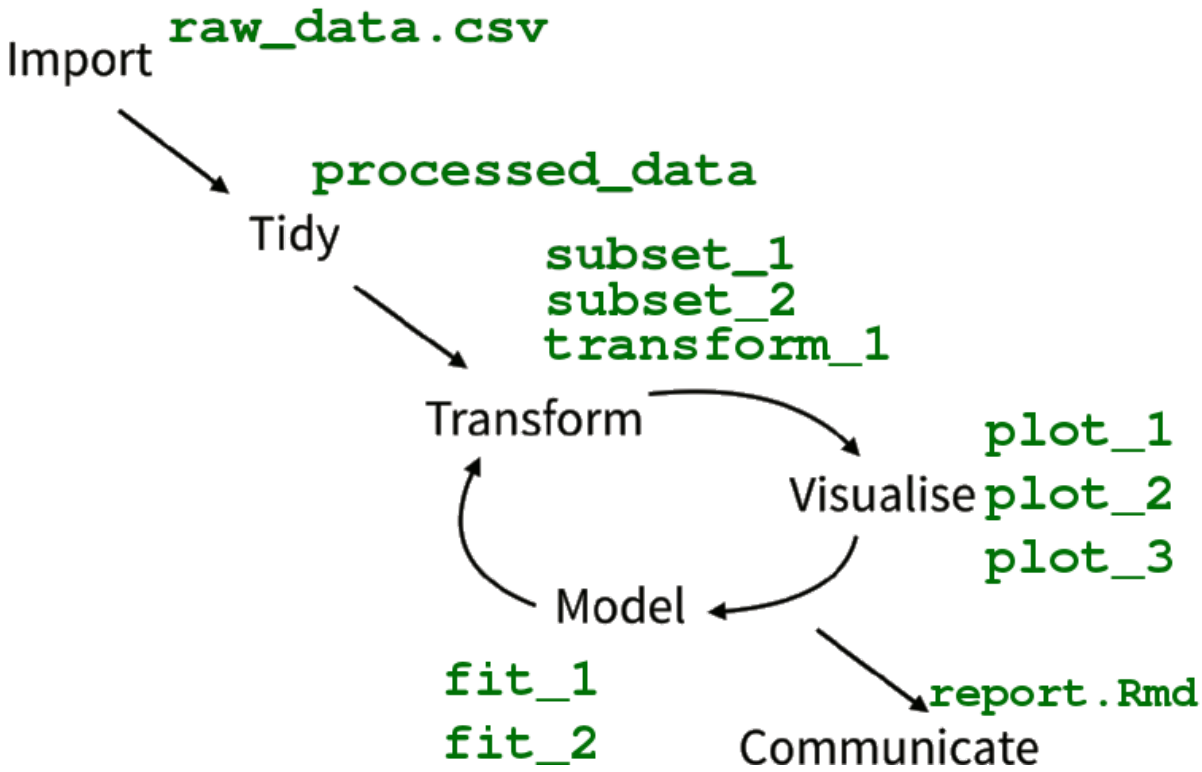
Xavier Laviron

December 6, 2018

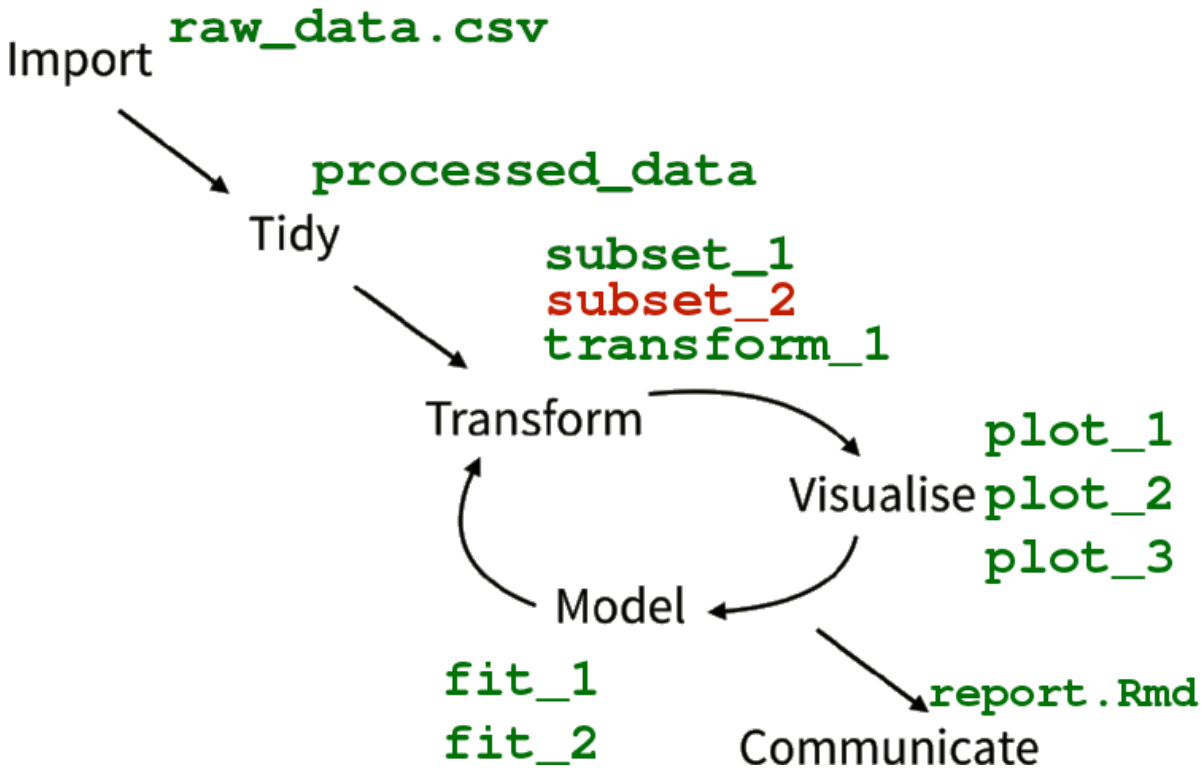
A data analyst's job



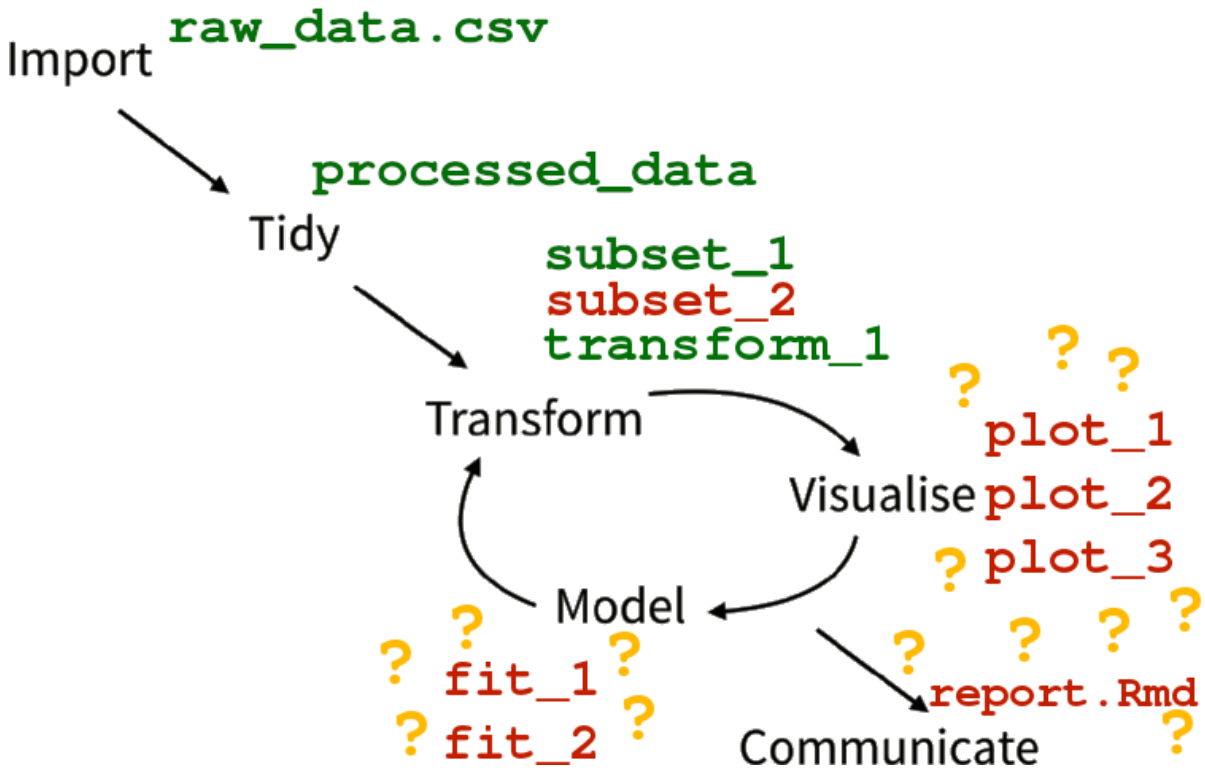
A data analyst's job



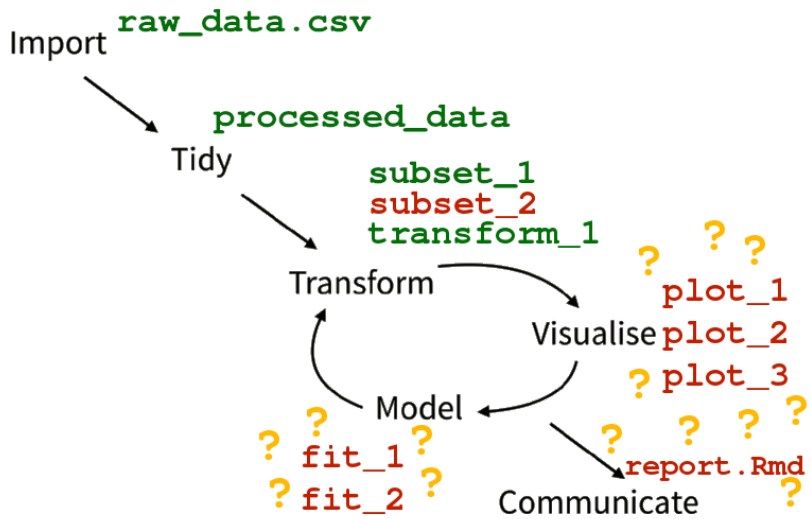
A data analyst's job



A data analyst's job



A data analyst's job



2 options:

- Run everything from scratch (simple, but can be too long...)
- Track the dependencies between your objects (boring, perfect job for a pipeline toolkit...)

The `drake` package is here to help you

Why use `drake`?

- Keeps track of dependencies in your code
- Keeps track of changes in your code
- Runs only what needs to be run, and skip the rest
- It has a cool name :-)

The `drake` package is here to help you

Why use `drake`?

- Keeps track of dependencies in your code
- Keeps track of changes in your code
- Runs only what needs to be run, and skip the rest
- It has a cool name :-)

In other words, 'drake' can save a lot of time!*

**more time for coffee breaks*

drake tracks changes in functions

Encapsulate your code in functions:

```
# Process the data
process_data <- function(raw.data) {
  raw.data[raw.data$Sepal.Length > 5, ]
}

# fit a model
fit_model <- function(data) {
  lm(Sepal.Length ~ Petal.Width + Species, data = data)
}

# create plots
create_plot <- function(data) {
  ggplot(data, aes(x = Petal.Width, fill = Species)) +
    geom_histogram()
}
```

The plan

The central piece of ‘drake‘: the workflow plan

The plan is a simple `data.frame` with two columns:

- `target`: the objects you want to build
- `command`: the functions to build them

The plan

The central piece of 'drake': the workflow plan

The plan is a simple `data.frame` with two columns:

- `target`: the objects you want to build
- `command`: the functions to build them

Different ways to create the plan:

- Like any `data.frame`: `data.frame()`, `expand.grid()`, ...
- With one of drake's helper functions: `drake_plan()`, `evaluate_plan()`, ...

The `drake_plan()` function

Usage:

```
drake_plan(target1 = command1, target2 = command2, ...)
```

The `drake_plan()` function

Usage:

```
drake_plan(target1 = command1, target2 = command2, ...)
```

```
my.plan <-
  drake_plan(raw.data  = read.csv(file_in("data/raw_data.csv")),

             proc.data = process_data(raw.data),

             plot      = create_plot(proc.data),

             model     = fit_model(proc.data),

             report    = render(input       = knitr_in("report.Rmd"),
                                output_file = file_out("report.pdf"),
                                quiet       = TRUE))
```

The `drake_plan()` function

```
print(my.plan)
```

```
## # A tibble: 5 x 2
##   target    command
## * <chr>     <chr>
## 1 raw.data  read.csv(file_in('data/raw_data.csv'))
## 2 proc.data process_data(raw.data)
## 3 plot      create_plot(proc.data)
## 4 model     fit_model(proc.data)
## 5 report    "render(input = knitr_in('report.Rmd'), output_file = file_ou~
```

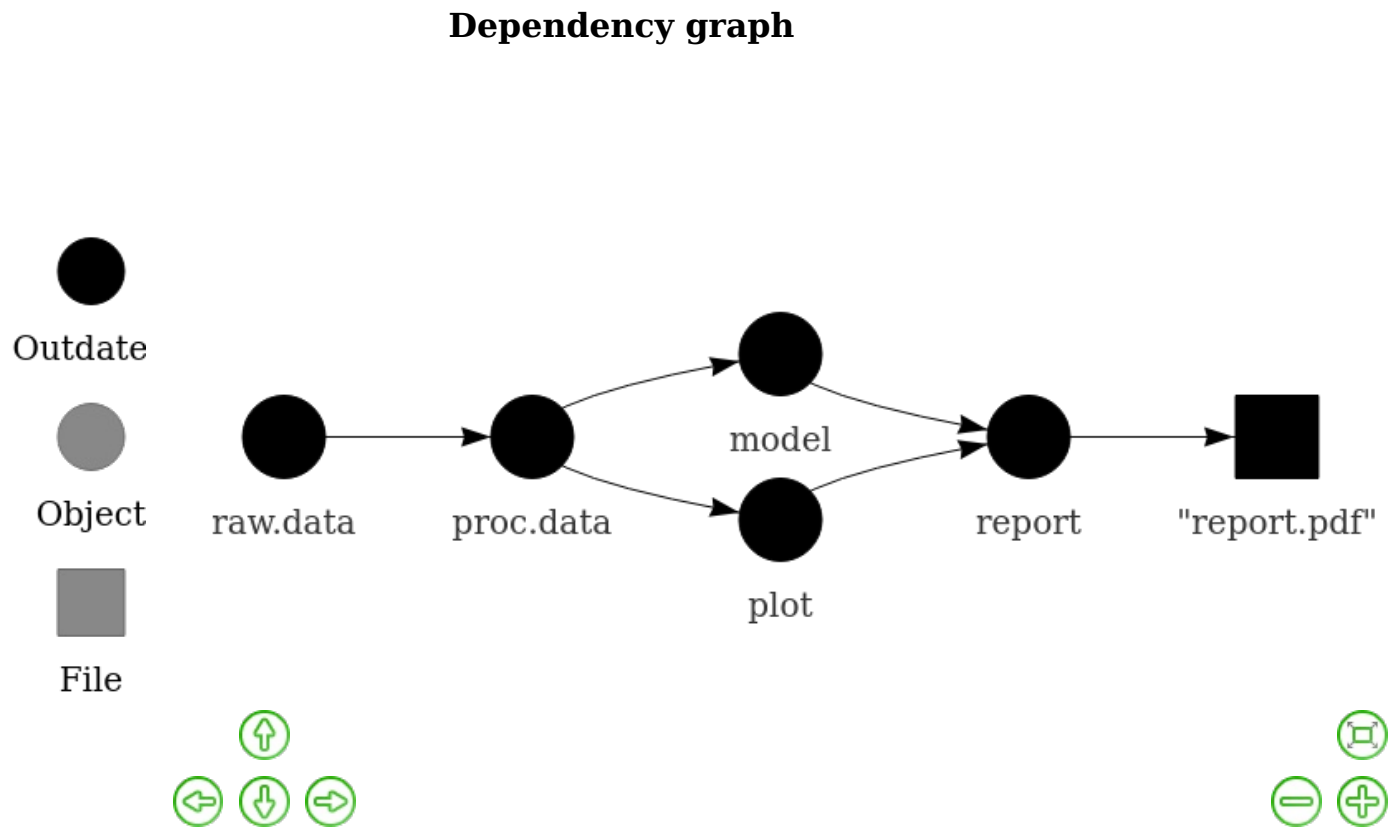
Files dependencies

Files are not tracked by `drake`, you have to declare them explicitly as dependencies:

- `file_in("some_data.csv")`: an input file
- `file_out("some_data.Rds")`: an output file
- `knitr_in("report.Rmd")`: an `rmarkdown` file, `drake` will scan it to find its dependencies

The dependency graph

```
vis_drake_graph(drake_config(my.plan), from = "raw.data")
```



The `make()` command

The central command of `drake`, runs everything that needs to run.

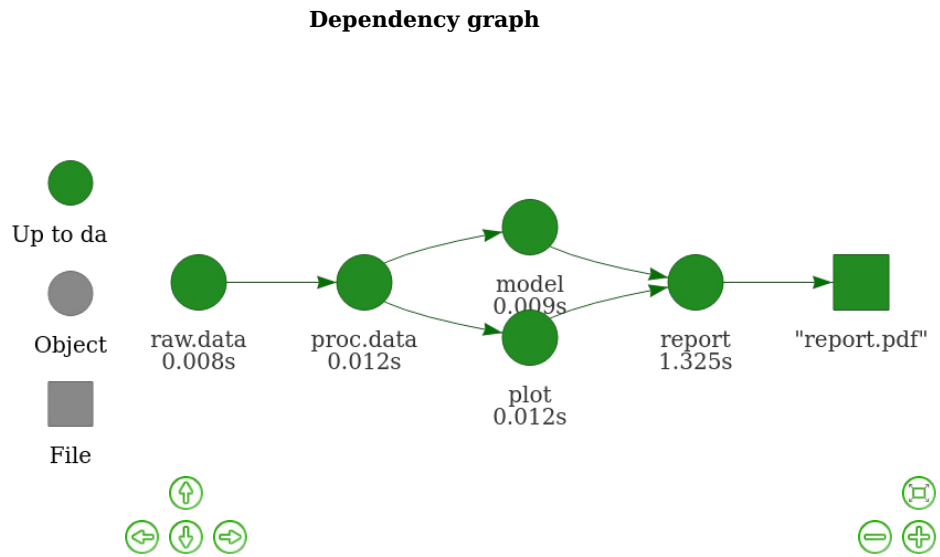
```
make(my.plan)
```

The `make()` command

The central command of `drake`, runs everything that needs to run.

```
make(my.plan)
```

```
vis_drake_graph(drake_config(my.plan), from = "raw.data")
```



Accessing the objects

All objects are stored in a hidden cache (`.drake/`). To access them:

```
loadd(model)
model <- readd(model)
```

Accessing the objects

All objects are stored in a hidden cache (`.drake/`). To access them:

```
loadd(model)
model <- readd(model)
```

```
print(readd(model))
```

```
##
## Call:
## lm(formula = Sepal.Length ~ Petal.Width + Species, data = data)
##
## Coefficients:
##       (Intercept)        Petal.Width  Speciesversicolor
##           5.13118            0.65802            -0.01955
##  Speciesvirginica
##           0.15373
```

An update in the code!

What happens if we modify a function?

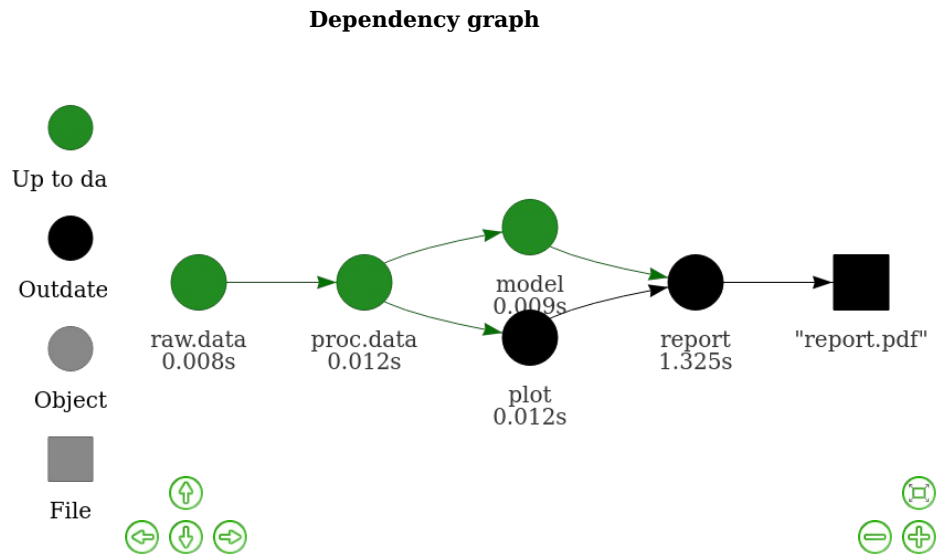
```
create_plot <- function(data) {
  ggplot(data, aes(x = Petal.Width, y = Sepal.Width, fill = Species)) +
    geom_point()
}
```

An update in the code!

What happens if we modify a function?

```
create_plot <- function(data) {
  ggplot(data, aes(x = Petal.Width, y = Sepal.Width, fill = Species)) +
    geom_point()
}
```

```
vis_drake_graph(drake_config(my.plan), from = "raw.data")
```



Other advantages

Reproducibilty

You have proof of what is done:

```
make(my.plan)
```

```
## All targets are already up to date.
```

Other advantages

Independant replication is made easy

- Your code is separated into functions: more readability and maintainability
- The plan allows an independent user to easily understand the analyses
- Restart everything from scratch easily:

```
outdated(drake_config(my.plan))
```

```
## character(0)
```

```
clean()
outdated(drake_config(my.plan))
```

```
## [1] "model"     "plot"      "proc.data" "raw.data"  "report"
```

Parallelization

- `drake` can manage multi-core computing (on a local machine or a HPC)
- Simply change the `jobs` argument of `make()`:

```
make(my.plan, jobs = 2)
```

Parallelization

- `drake` can manage multi-core computing (on a local machine or a HPC)
- Simply change the `jobs` argument of `make()`:

```
make(my.plan, jobs = 2)
```

- `drake` will automatically know which targets can be run in parallel and which cannot

Ressources

To go further

`https://github.com/ropensci/drake`

- Online documentation
- Cheatsheet
- FAQ

The package is in active development and there are a lot of other functionnalities

Exercices

There exists a bunch of built-in examples, you can list them with:

```
drake_examples()
```

And then load one with:

```
drake_example("example_name")
```

This will create a directory with all the necessary files, that you can open in the IDE of your choice (Rstudio, vim, ...).

Exercice: The basic example

The most accessible example for beginners

```
drake_example("main")
```

Exercice: The `mtcars` example

This chapter is a walkthrough of `drake`'s main functionality based on the mtcars example. It sets up the project and runs it repeatedly to demonstrate `drake`'s most important functionality.

```
drake_example("mtcars")
```

Exercice: An analysis of R package download trends

This example explores R package download trends using the `cranlogs` package, and it shows how `drake`'s custom triggers can help with workflows with remote data sources.

```
drake_example("packages")
```