Reproducible workflows at scale with drake

Will Landau

Large data science workflows

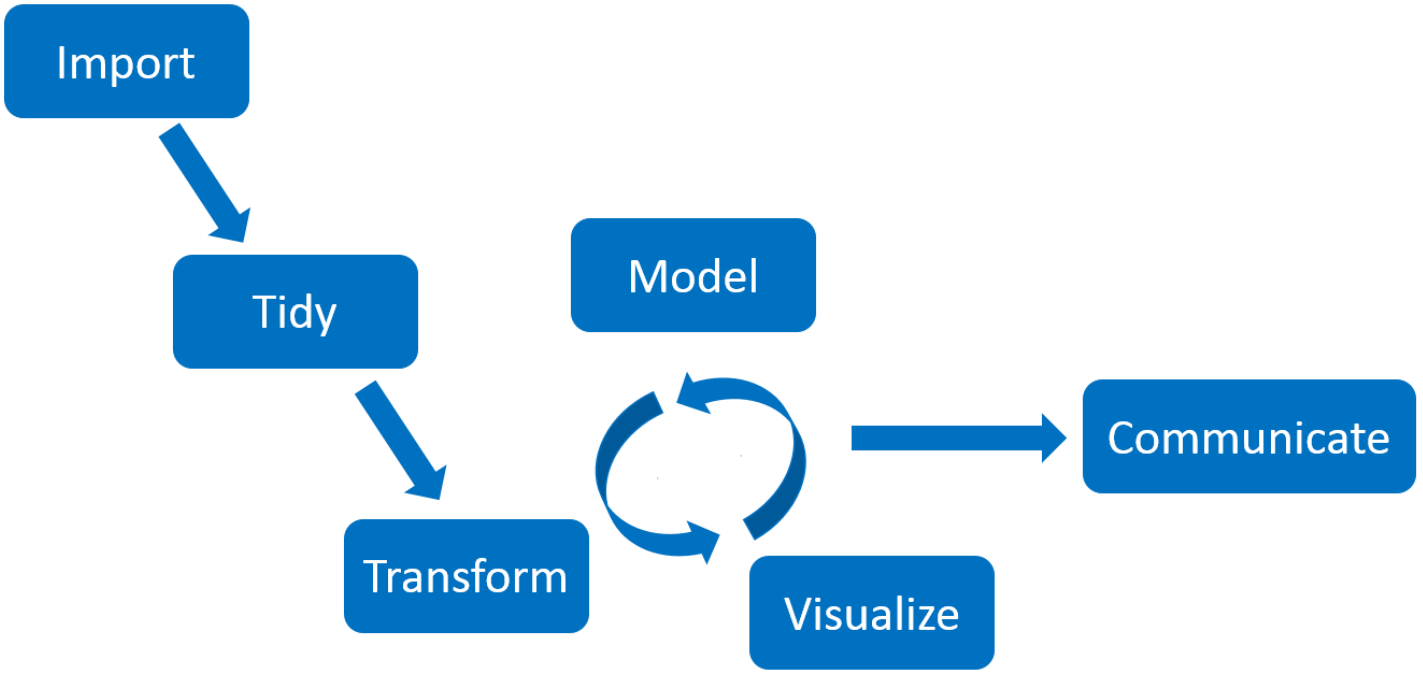
- Struggles

    1. Long runtimes.
    2. Many tasks.
    3. Interconnected tasks.

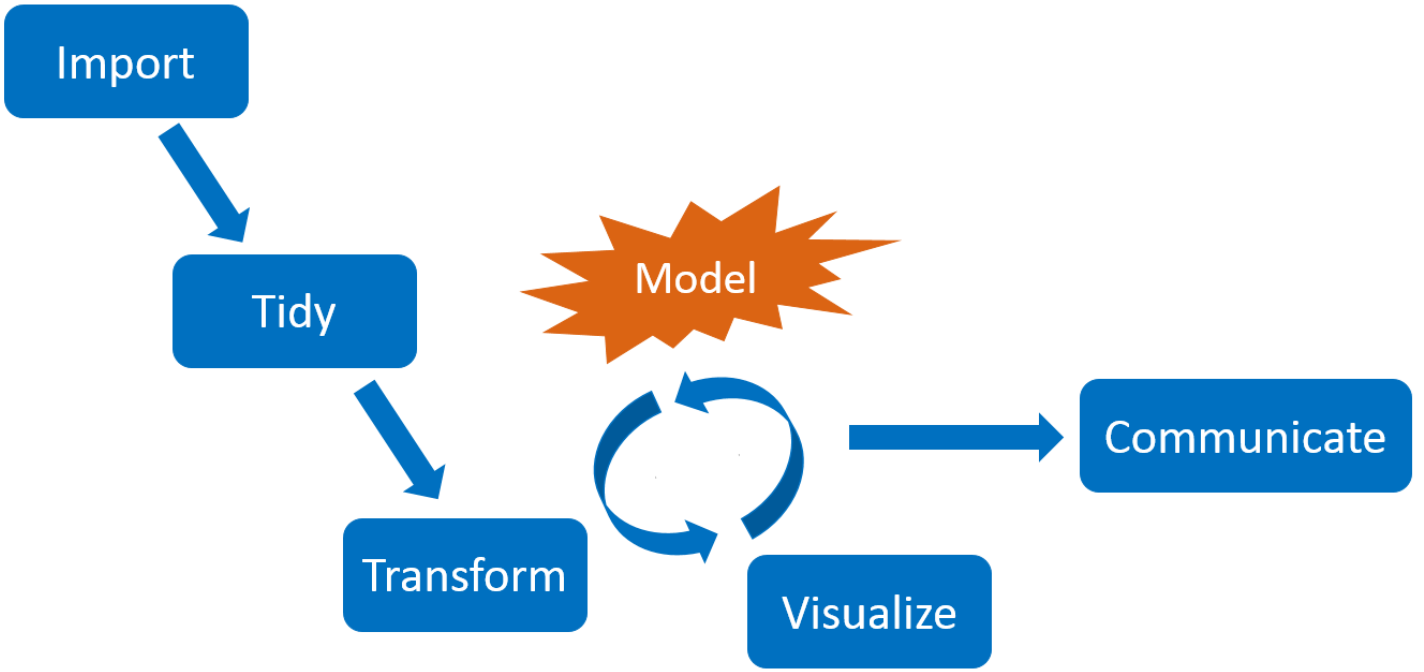
- Examples

    - Deep learning.
    - Classical machine learning.
    - Bayesian data analysis via Markov chain Monte Carlo.
    - Spatial data analysis.
    - Clinical trial modeling and simulation.
    - Subgroup identification.
    - Graph-based multiple comparison procedures.
    - Genomics pipelines.
    - PK/PD modeling.

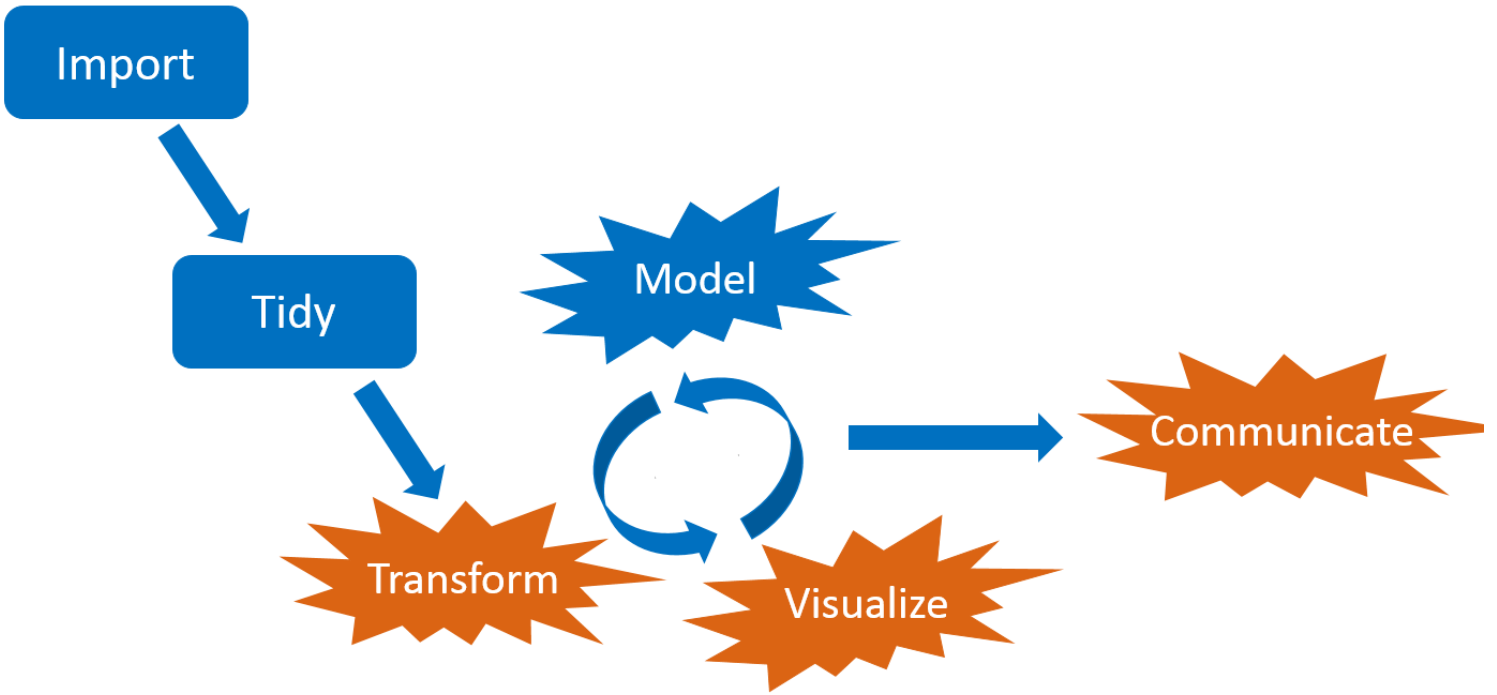
Interconnected tasks



When you change something...



...the downstream output is **no longer valid**.



Do you rerun **everything** from scratch?

- Not if you deal with long runtimes!

https://openclipart.org/detail/275842/sisyphus-overcoming-silhouette

Do you pick and choose what to update?

- Messy.
- Prone to human error.
- Not reproducible.

https://openclipart.org/detail/216179/messy-desk

Solution: pipeline tools

Scale up the work you need.

Skip the work you don't.

See evidence of reproducibility.

- Tons exist already: github.com/pditommaso/awesome-pipeline.
- Most are language-agnostic or designed for Python or the shell.

What distinguishes `drake`?

- Aggressively designed for R.

1. Think **functions**, not script files.
2. Think **variables**, not output files.
3. Think **data frames**, not `Makefiles`.

- `drake` borrows (1) and (2) from the `remake` package by Rich FitzJohn.
- `remake` is no longer under development.
- `drake` tries to extend `remake`'s ideas further and handle larger projects.

Example: a deep learning workflow

- Goal: predict customers who cancel their subscriptions with a telecom company.
- Data: IBM Watson Telco Customer Churn dataset.
- Workflow principles generalize to other industries.

https://openclipart.org/detail/90739/newplus, https://github.com/rstudio/keras

❌ Let's move beyond numbered scripts.

```
run_everything.R
R/
├── 01-data.R
├── 02-munge.R
├── 03-model.R
├── 04-results.R
└── 05-plot.R
data/
└── customer_churn.csv
output/
├── model_relu.h5
├── model_sigmoid.h5
├── confusion_matrix.rds
└── metrics_plot.png
```

❌ Why not numbered scripts?

- The planning and the execution happen at the same time.
- Too cumbersome, ad hoc, and tangled for ambitious projects.

```
# 02-munge.R
library(recipes) # Package dependencies scattered across scripts.

rec <- data %>% # Single-use code, difficult to test.
  training() %>%
  recipe(Churn ~ .) %>%
  step_rm(customerID) %>%
  step_naomit(all_outcomes(), all_predictors()) %>%
  step_discretize(tenure, options = list(cuts = 6)) %>%
  step_log(TotalCharges) %>%
  step_mutate(Churn = ifelse(Churn == "Yes", 1, 0)) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_center(all_predictors(), -all_outcomes()) %>%
  step_scale(all_predictors(), -all_outcomes()) %>%
  prep()

saveRDS(rec, "recipe.rds") # Final output scattered across code.
```

✓ Instead, embrace **functions!**

- A function is a reusable command that accepts one or more inputs and returns a single output.
- It's a piece of custom shorthand for a single idea.

```
add_things <- function(argument1, argument2) {
  argument1 + argument2
}

add_things(1, 2)
## [1] 3

add_things(c(3, 4), c(5, 6))
## [1]  8 10
```

Why use functions?

1. Clarity: break down complicated ideas into manageable pieces.

2. Use R as intended.

> - Everything that exists is an object.
> - Everything that happens is a function call.
>
> John Chambers

3. Reuse: define once, run wherever.

Functions in a workflow

```
make.R
R/
├── packages.R
├── functions.R
└── plan.R
data/
└── customer_churn.csv
.drake/ # drake's cache
└──     # Output automatically appears here.
```

Functions in a workflow

```r
# packages.R: all package dependencies
library(recipes)
# other packages...
```

```r
# functions.R: pure reusable code
prepare_recipe <- function(data) {
  data %>%
    training() %>%
    recipe(Churn ~ .) %>%
    step_rm(customerID) %>%
    step_naomit(all_outcomes(), all_predictors()) %>%
    step_discretize(tenure, options = list(cuts = 6)) %>%
    step_log(TotalCharges) %>%
    step_mutate(Churn = ifelse(Churn == "Yes", 1, 0)) %>%
    step_dummy(all_nominal(), -all_outcomes()) %>%
    step_center(all_predictors(), -all_outcomes()) %>%
    step_scale(all_predictors(), -all_outcomes()) %>%
    prep()
}
# other functions...
```

Functions in a workflow

```
# later in functions.R...

run_everything <- function() {
  data <- read_csv(file_in("data/customer_churn.csv"), col_types = cc
    initial_split(prop = 0.3)
  saveRDS(data, "output/data.rds")

  rec <- prepare_recipe(data) # Call your other functions.
  saveRDS(rec, "output/rec.rds")

  model_relu <- train_model(rec, act1 = "relu")
  save_model_hdf5(model_relu, "output/model_relu.h5")
  # more models...

  conf_sigmoid <- confusion_matrix(data, rec, model_sigmoid)
  saveRDS(conf_sigmoid, "output/conf_sigmoid.rds")
  # more confusion matrices...

  metrics <- compare_models(conf_relu, conf_sigmoid)
  saveRDS(metrics, "output/metrics.rds")
}
```

Conduct your analysis with your **functions**.

```
# run_everything.R
source("R/packages.R")
source("R/functions.R")
run_everything()
```

But we can still do better...

- Avoid rerunning all the computation every time.
- Avoid micromanaging output files.

https://publicdomainvectors.org/en/free-clipart/Golden-magic-lamp/61683.html

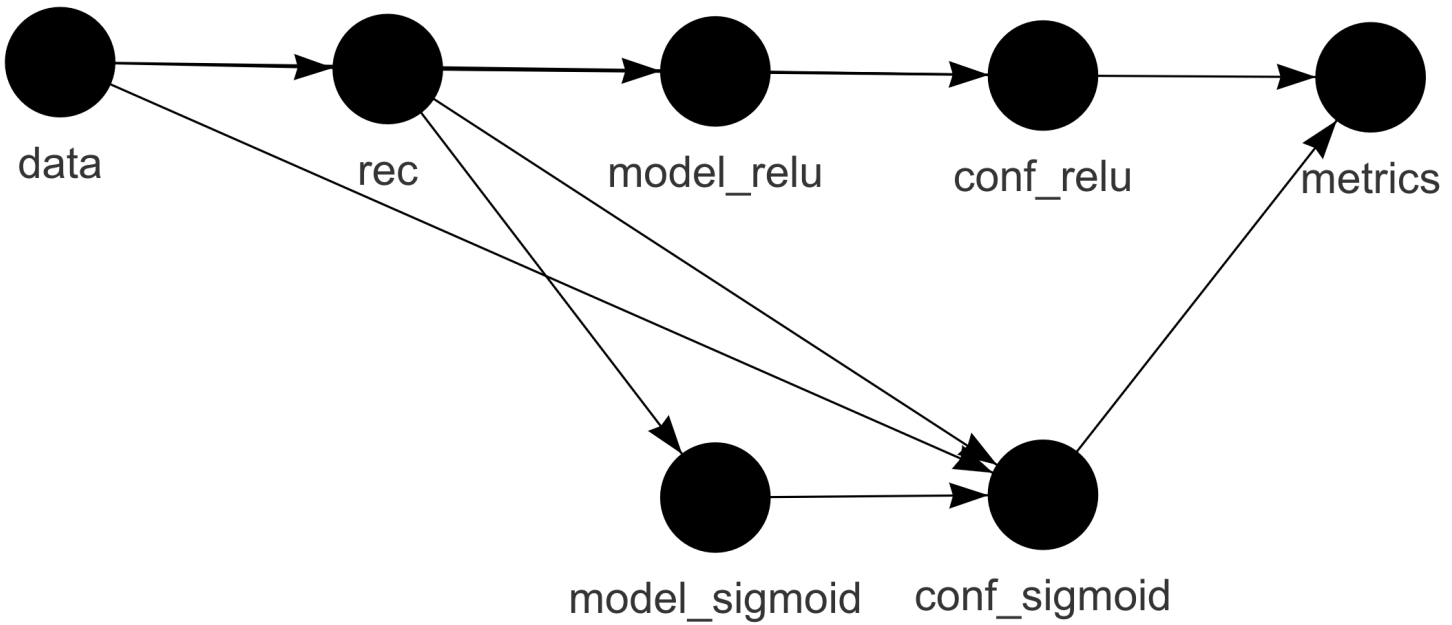
Enter drake! Define a **plan**.

```
plan <- drake_plan(
  rec = prepare_recipe(data), # Use your functions.
  model = target(
    train_model(rec, act1 = act),
    format = "keras",
    transform = map(act = c("relu", "sigmoid"))
  ),
  conf = target(
    confusion_matrix(data, rec, model),
    transform = map(model, .id = act)
  ),
  metrics = target(
    compare_models(conf),
    transform = combine(conf)
  ),
  data = read_csv(                        # flexible target order,
    file_in("data/customer_churn.csv"), # flexible commands
    col_types = cols()
  ) %>%
    initial_split(prop = 0.3)
)
```

The plan is a data frame of skippable tasks.

```
plan
## # A tibble: 7 x 3
##   target      command
##   <chr>       <expr>
## 1 rec         prepare_recipe(data)
## 2 model_relu  train_model(rec, act1 = "relu")
## 3 model_sigm… train_model(rec, act1 = "sigmoid")
## 4 conf_relu   confusion_matrix(data, rec, model_relu)
## 5 conf_sigmo… confusion_matrix(data, rec, model_sigmoid)
## 6 metrics     compare_models(conf_relu, conf_sigmoid)
## 7 data        read_csv(file_in("data/customer_churn.csv"), col_typ
```

The workflow



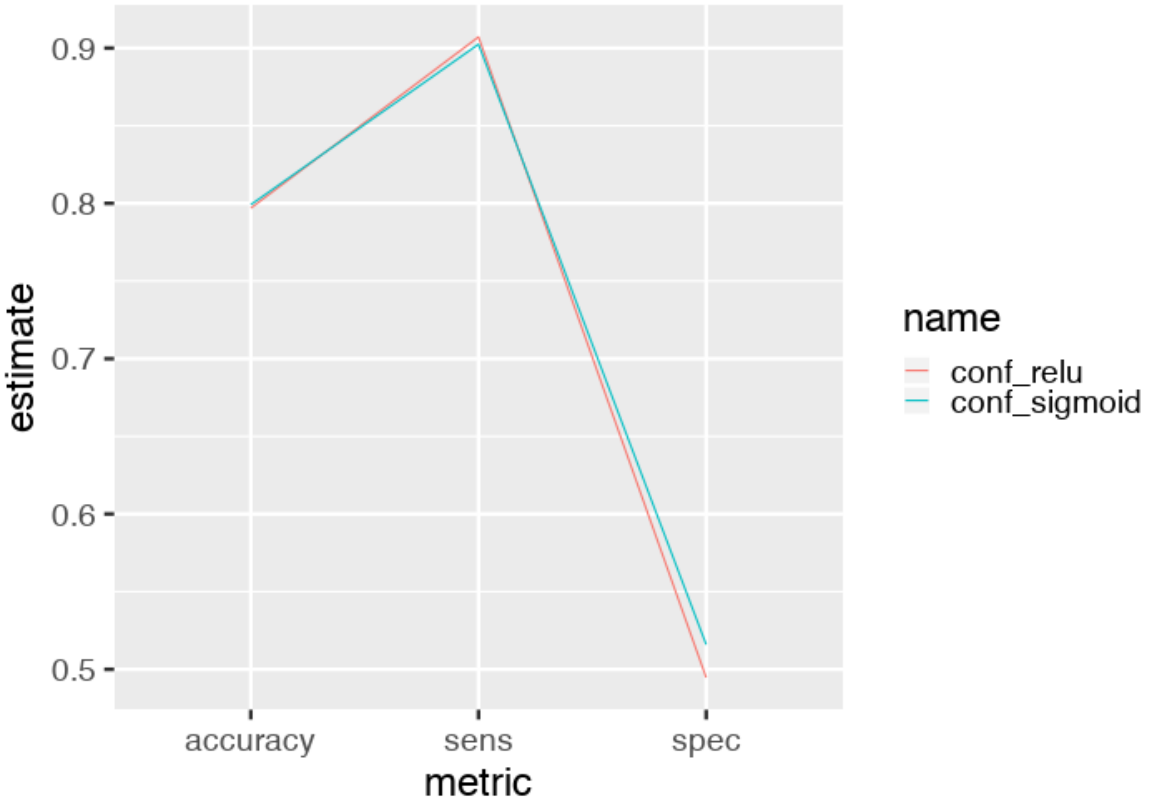
Run the project in make.R.

```
# like run_everything.R...
source("R/packages.R")
source("R/functions.R")
source("R/plan.R")

make(plan)
## target data
## target rec
## target model_relu
## target model_sigmoid
## target conf_relu
## target conf_sigmoid
## target metrics
```

Compare models.

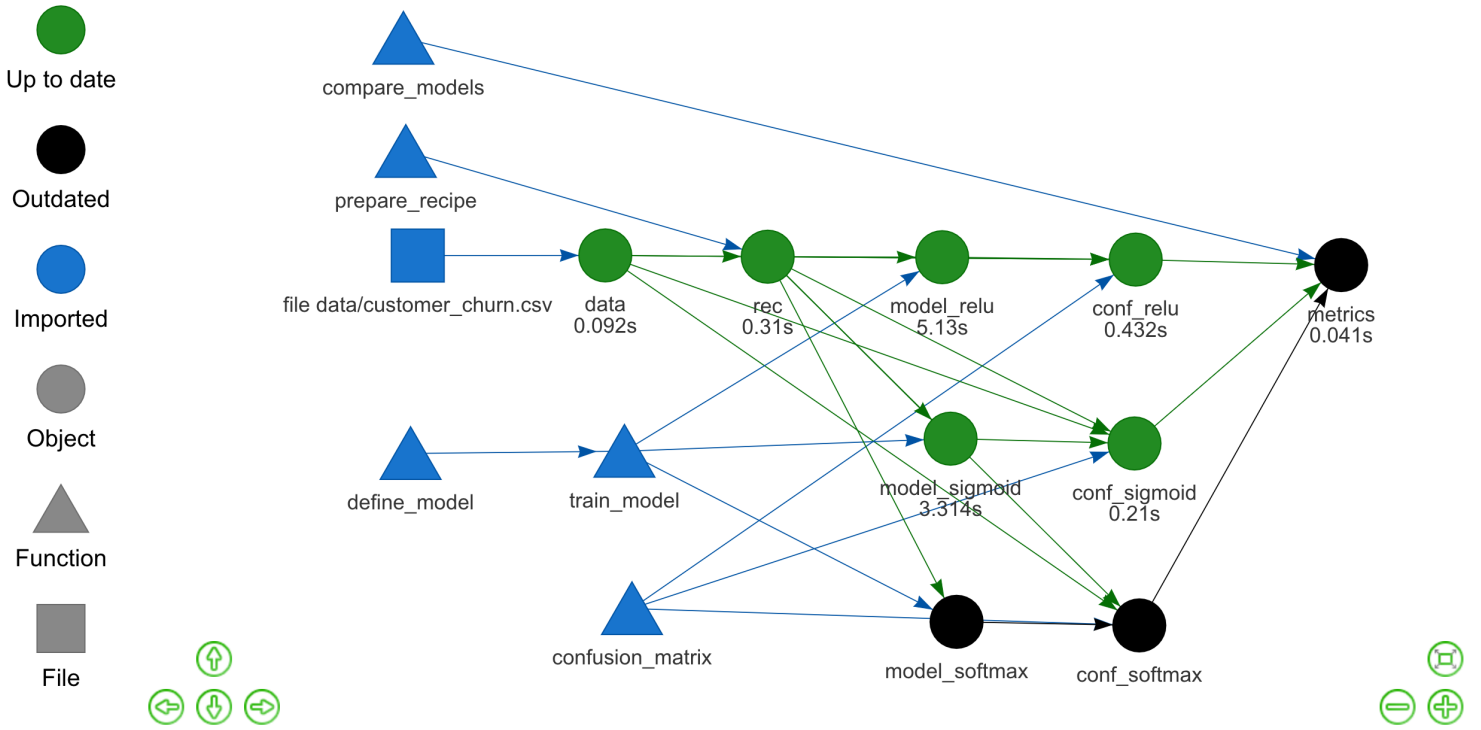
```
readd(metrics) # See also loadd()
```



Add a new model.

```
plan <- drake_plan(
  rec = prepare_recipe(data),
  model = target(
    train_model(rec, act1 = act),
    format = "keras",
    transform = map(act = c("relu", "sigmoid", "softmax"))
  ),
  conf = target(
    confusion_matrix(data, rec, model),
    transform = map(model, .id = act)
  ),
  metrics = target(
    compare_models(conf),
    transform = combine(conf)
  ),
  data = read_csv(
    file_in("data/customer_churn.csv"),
    col_types = cols()
  ) %>%
    initial_split(prop = 0.3)
)
```

vis_drake_graph()



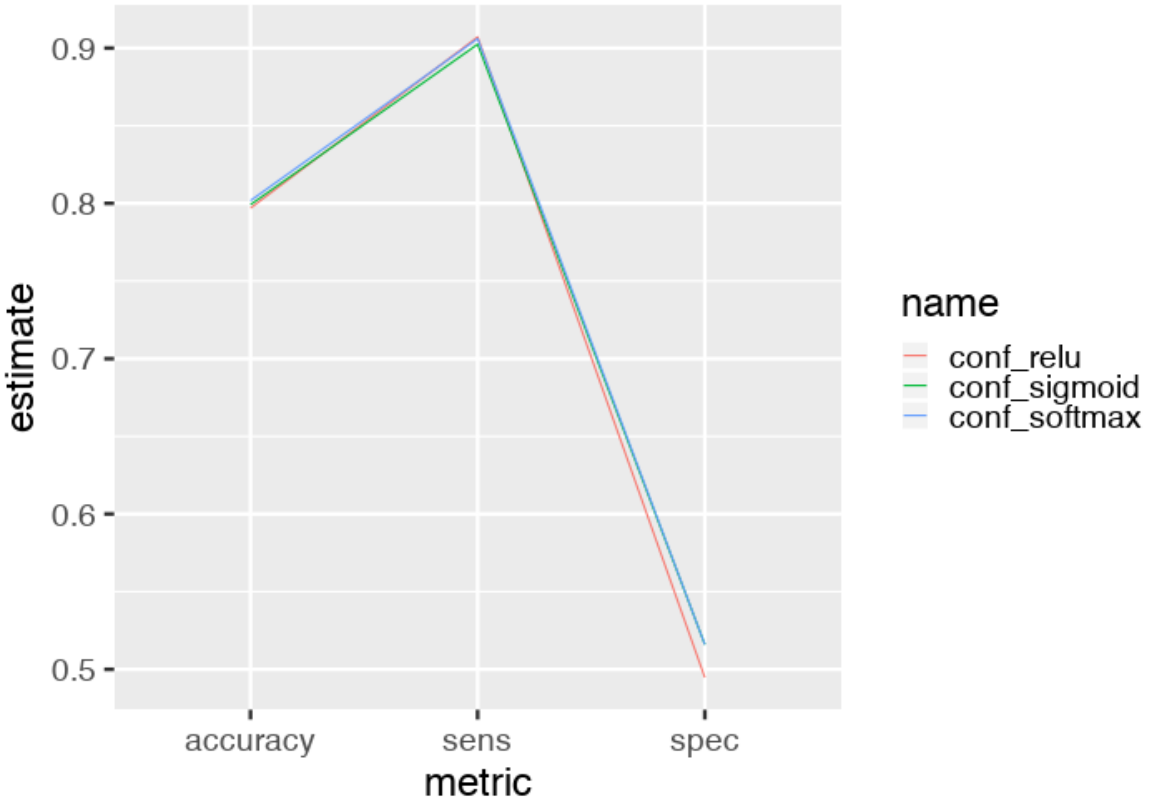
Refresh the results in make.R.

```
source("R/packages.R")
source("R/functions.R")
source("R/plan.R") # modified

make(plan)
## target model_softmax
## target conf_softmax
## target metrics
```

Compare models.

```
readd(metrics)
```



Evidence of reproducibility

```
source("R/packages.R")
source("R/functions.R")
source("R/plan.R")

make(plan)
## All targets are already up to date.
```

- See also `outdated()`.

Efficient data formats

- Increased speed and reduced memory consumption.

```
library(drake)
n <- 1e8 # Each target is 1.6 GB in memory.
plan <- drake_plan(
  data_fst = target(
    data.frame(x = runif(n), y = runif(n)),
    format = "fst"
  ),
  data_old = data.frame(x = runif(n), y = runif(n))
)
make(plan)
#> target data_fst
#> target data_old
build_times(type = "build")
#> # A tibble: 2 x 4
#>   target   elapsed              user                 system
#>   <chr>    <Duration>           <Duration>           <Duration>
#> 1 data_fst 13.93s               37.562s              7.954s
#> 2 data_old 184s (~3.07 minutes) 177s (~2.95 minutes) 4.157s
```

History and provenance

```
drake_history()
## # A tibble: 10 x 10
##    target  current built  exists hash  command     seed runtime  p
##    <chr>   <lgl>   <chr>  <lgl>  <chr> <chr>      <int>   <dbl> <c
##  1 conf_r… TRUE    2019-… TRUE   a946… confusio… 4.05e8  0.231   N
##  2 conf_s… TRUE    2019-… TRUE   b666… confusio… 1.93e9  0.233   N
##  3 conf_s… TRUE    2019-… TRUE   8df0… confusio… 1.80e9  0.234   N
##  4 data    TRUE    2019-… TRUE   ca84… "read_cs… 1.29e9  0.051
##  5 metrics FALSE   2019-… TRUE   3109… compare_… 1.21e9  0.0250  N
##  6 metrics TRUE    2019-… TRUE   1c48… compare_… 1.21e9  0.024   N
##  7 model_… TRUE    2019-… TRUE   9ef3… "train_m… 1.47e9  7.98    N
##  8 model_… TRUE    2019-… TRUE   9c0c… "train_m… 1.26e9  3.54    N
##  9 model_… TRUE    2019-… TRUE   82da… "train_m… 8.05e8  4.29    N
## 10 rec     TRUE    2019-… TRUE   eae9… prepare_… 6.29e8  0.190   N
```

Reproducible data recovery

```
clean() # Oops!

start <- proc.time()
make(plan, recover = TRUE)
## recover data
## recover rec
## recover model_relu
## recover model_sigmoid
## recover model_softmax
## recover conf_relu
## recover conf_sigmoid
## recover conf_softmax
## recover metrics

proc.time() - start
##    user  system elapsed
##   0.109   0.048   0.333
```

- Details + how to rename a target: https://ropenscilabs.github.io/drake-manual/walkthrough.html#reproducible-data-recovery-and-renaming

Dependency-aware high-performance computing

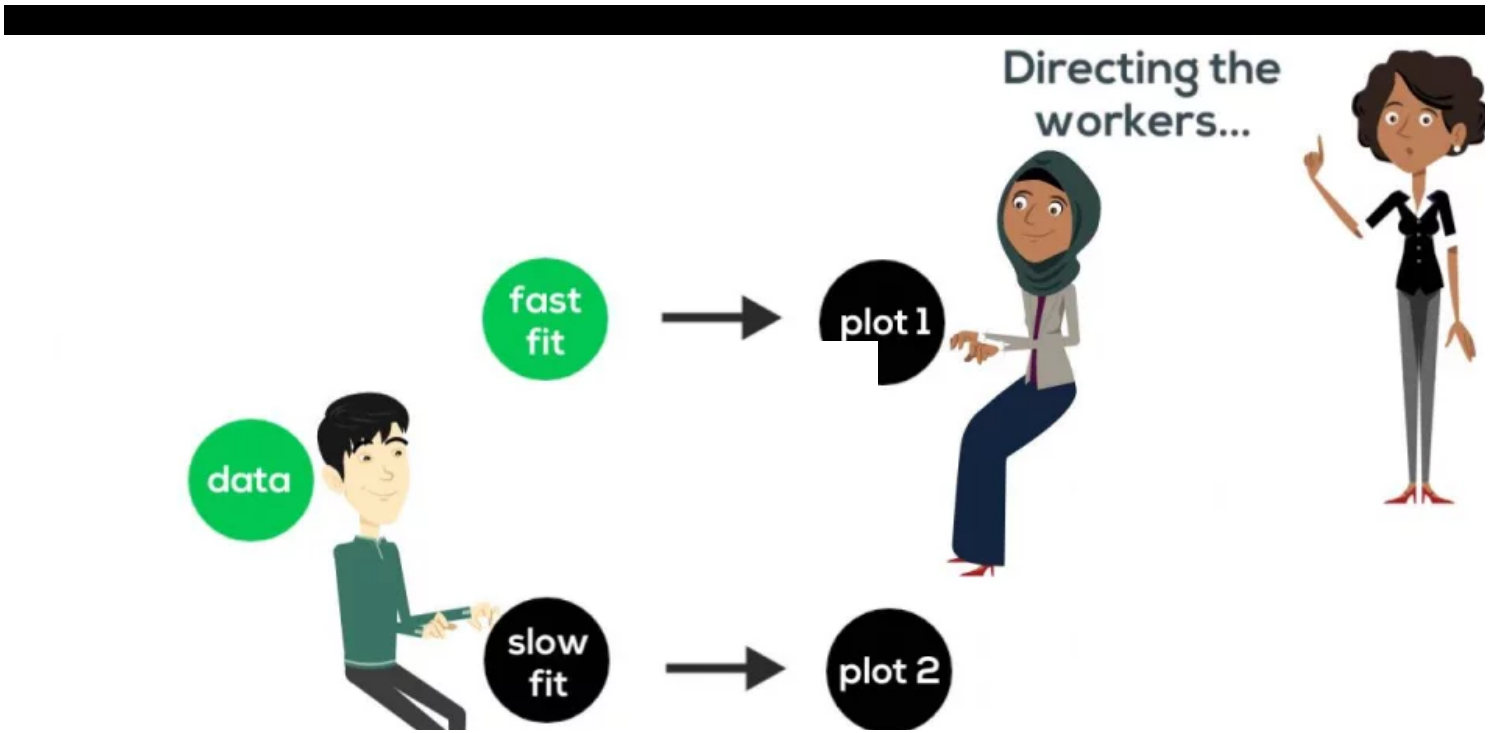
- Just a little configuration...

```
# template file with configuration
drake_hpc_template_file("slurm_clustermq.tmpl")

# Use SLURM resource manager with the template.
options(
  clustermq.scheduler = "slurm",
  clustermq.template = "slurm_clustermq.tmpl"
)

# make() is the basically the same.
make(plan, jobs = 2, parallelism = "clustermq")
```

Dependency-aware high-performance computing



Resources

- Get `drake`:

```
install.packages("drake")
```

- Example code from these slides:

```
drake::drake_example("customer-churn")
```

- Workshop materials:

```
remotes::install_github("wlandau/learndrake")
```

Links

- Development repository: https://github.com/ropensci/drake
- Full user manual https://ropenscilabs.github.io/drake-manual
- Reference website: https://docs.ropensci.org/drake
- Hands-on workshop: https://github.com/wlandau/learndrake
- Code examples: https://github.com/wlandau/drake-examples
- Discuss at rOpenSci.org: https://discuss.ropensci.org

rOpenSci use cases

- Use `drake`? Share your use case at https://ropensci.org/usecases.

Thanks

- Edgar Ruiz
- example code

- Matt Dancho
- blog post

Thanks

- Maëlle Salmon
- Ben Marwick
- Julia Lowndes
- Peter Slaughter
- Jenny Bryan
- Rich FitzJohn
- Stefanie Butland

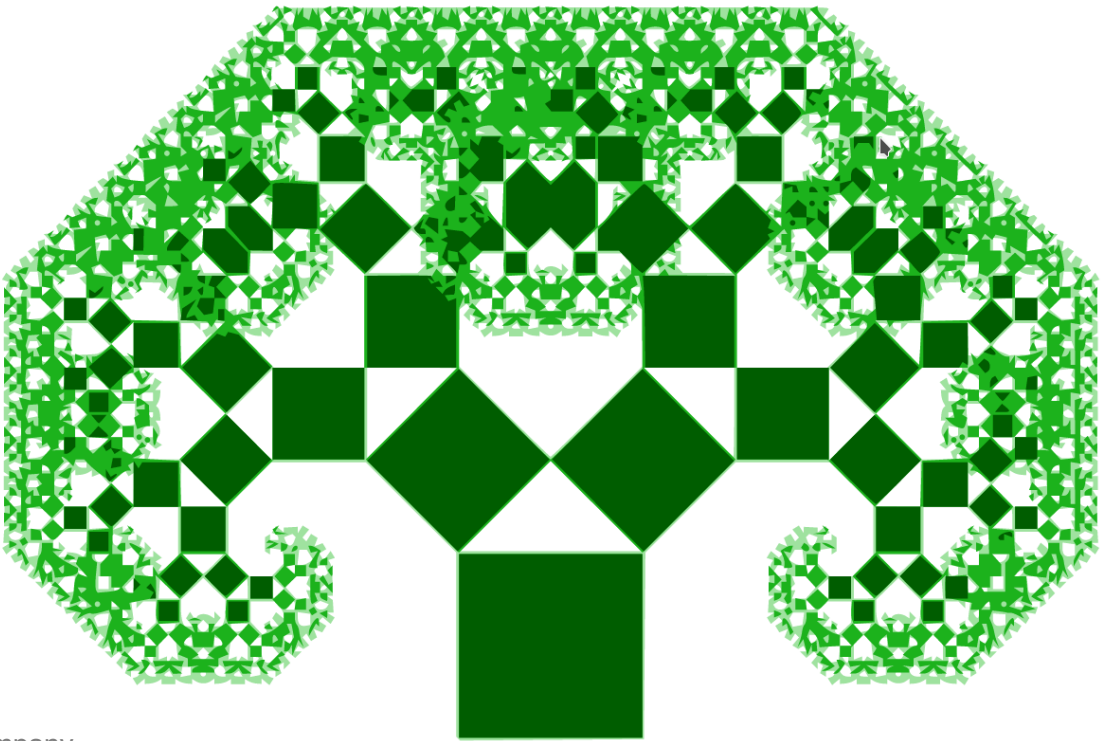
- Jarad Niemi
- Kirill Müller
- Henrik Bengtsson
- Michael Schubert
- Kendon Bell
- Miles McBain
- Patrick Schratz
- Alex Axthelm
- Jasper Clarkberg
- Tiernan Martin
- Ben Listyg
- TJ Mahr
- Ben Bond-Lamberty
- Tim Mastny
- Bill Denney
- Amanda Dobbyn
- Daniel Falster
- Rainer Krug
- Brianna McHorse
- Chan-Yub Park

A riddle!

- From a math PhD oral exam:

  > Define an example of a nontrivial function.

- Hint: the best answers do not even come from math or computing!



https://publicdomainvectors.org/en/free-clipart/Pythagoras-tree/58775.html