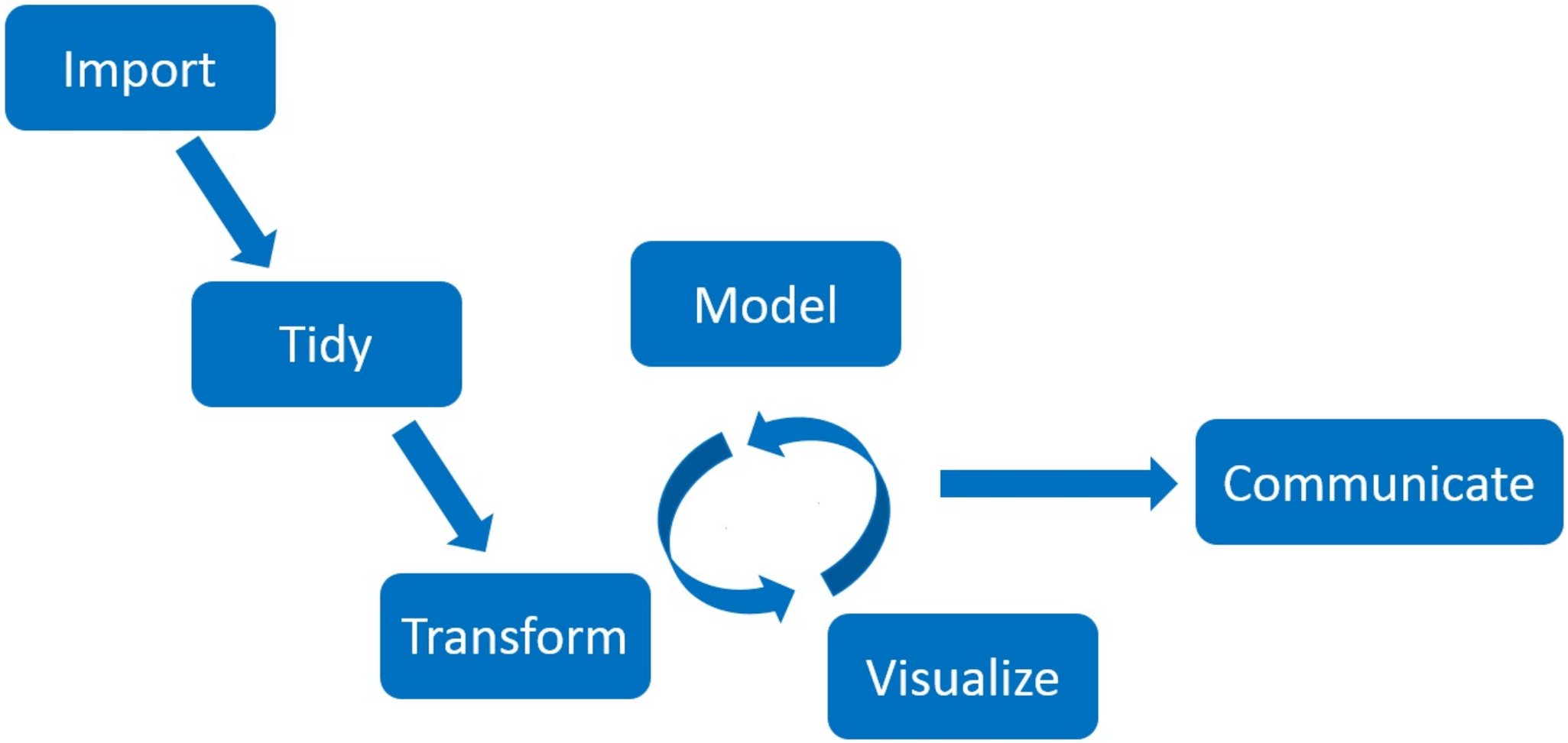
Reproducible workflows at scale with drake

Will Landau

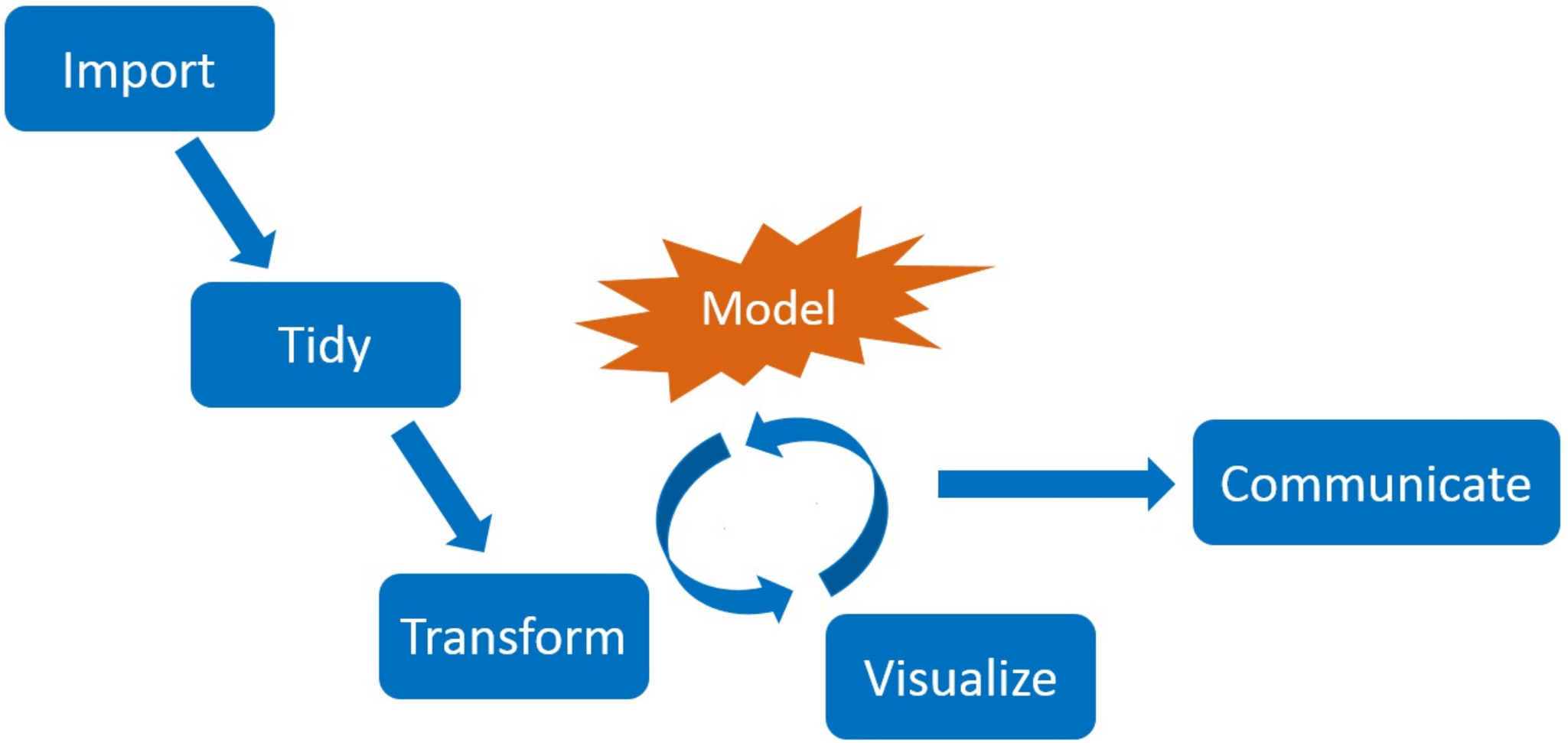
Large data science workflows

- Struggles
    1. Long runtimes.
    2. Many tasks.
    3. Interconnected tasks.
- Examples
    - Deep learning
    - Classical machine learning.
    - Bayesian computation via Markov chain Monte Carlo (`rjags`, `rstan`, etc.)
    - Spatial data analysis.
    - Clinical trial modeling and simulation.
    - Subgroup identification.
    - Graph-based multiple comparison procedures.
    - Genomics pipelines.
    - PK/PD modeling (`mrgsolve`, `nlmixr`, etc.)

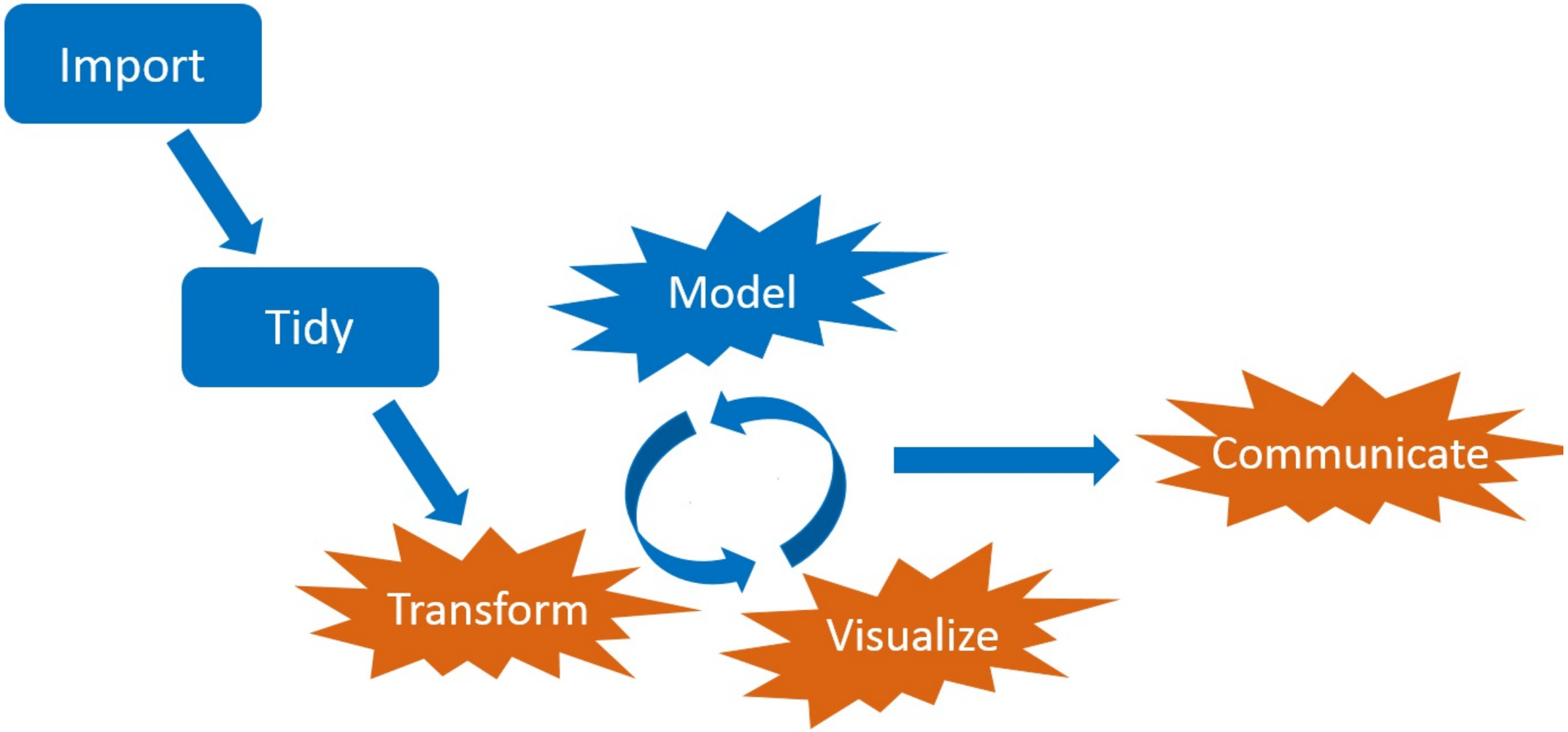
Interconnected tasks



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...the downstream output is no longer valid.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

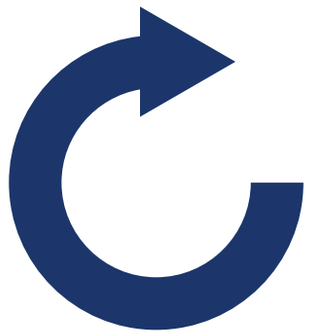
- Messy.
- Prone to human error.
- Not reproducible.

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!

- Separate the planning from the execution.
- Clarity: break down complicated ideas into manageable pieces.
- Reuse: define once, run whenever: testing, development, etc.

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

✓ Write function-oriented R scripts.

```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...
```

Conduct your analysis with your functions.

```
# run_everything.R
source("R/packages.R")
source("R/functions.R")
data <- read_csv(file_in("data/customer_churn.csv"), col_types = cols()) %>%
  initial_split(prop = 0.3)
saveRDS(data, "output/data.rds")
rec <- prepare_recipe(data) # Use your functions.
saveRDS(rec, "output/rec.rds")
model_relu <- train_model(rec, act1 = "relu")
save_model_hdf5(model_relu, "output/model_relu.h5")
model_sigmoid <- train_model(rec, act1 = "sigmoid")
save_model_hdf5(model_sigmoid, "output/model_sigmoid.h5")
conf_relu <- confusion_matrix(data, rec, model_relu)
saveRDS(conf_relu, "output/conf_relu.rds")
conf_sigmoid <- confusion_matrix(data, rec, model_sigmoid)
saveRDS(conf_sigmoid, "output/conf_sigmoid.rds")
metrics <- compare_models(conf_relu, conf_sigmoid)
saveRDS(metrics, "output/metrics.rds")
```

But we can still do better...

- Avoid rerunning everything every time.
- Avoid micromanaging files. No more `saveRDS()`, `save_model_hdf5()`, etc.

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

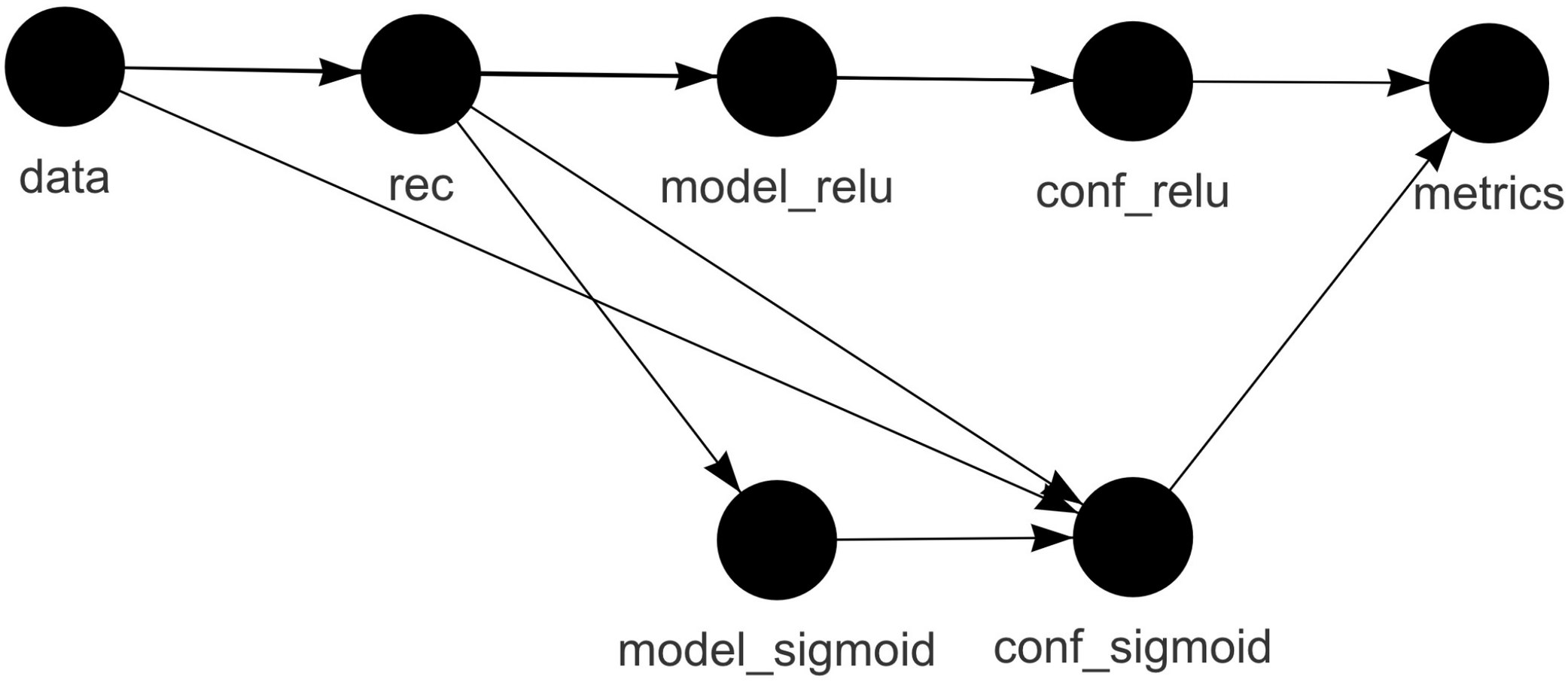
Enter drake! Define a plan.

```
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                                                  forma
##   <chr>       <expr>                                                   <chr
## 1 rec         prepare_recipe(data)                                   … <NA>
## 2 model_relu  train_model(rec, act1 = "relu")                        … keras
## 3 model_sigm… train_model(rec, act1 = "sigmoid")                     … keras
## 4 conf_relu   confusion_matrix(data, rec, model_relu)                … <NA>
## 5 conf_sigmo… confusion_matrix(data, rec, model_sigmoid)             … <NA>
## 6 metrics     compare_models(conf_relu, conf_sigmoid)                … <NA>
## 7 data        read_csv(file_in("data/customer_churn.csv"), col_type… <NA>
```

The workflow



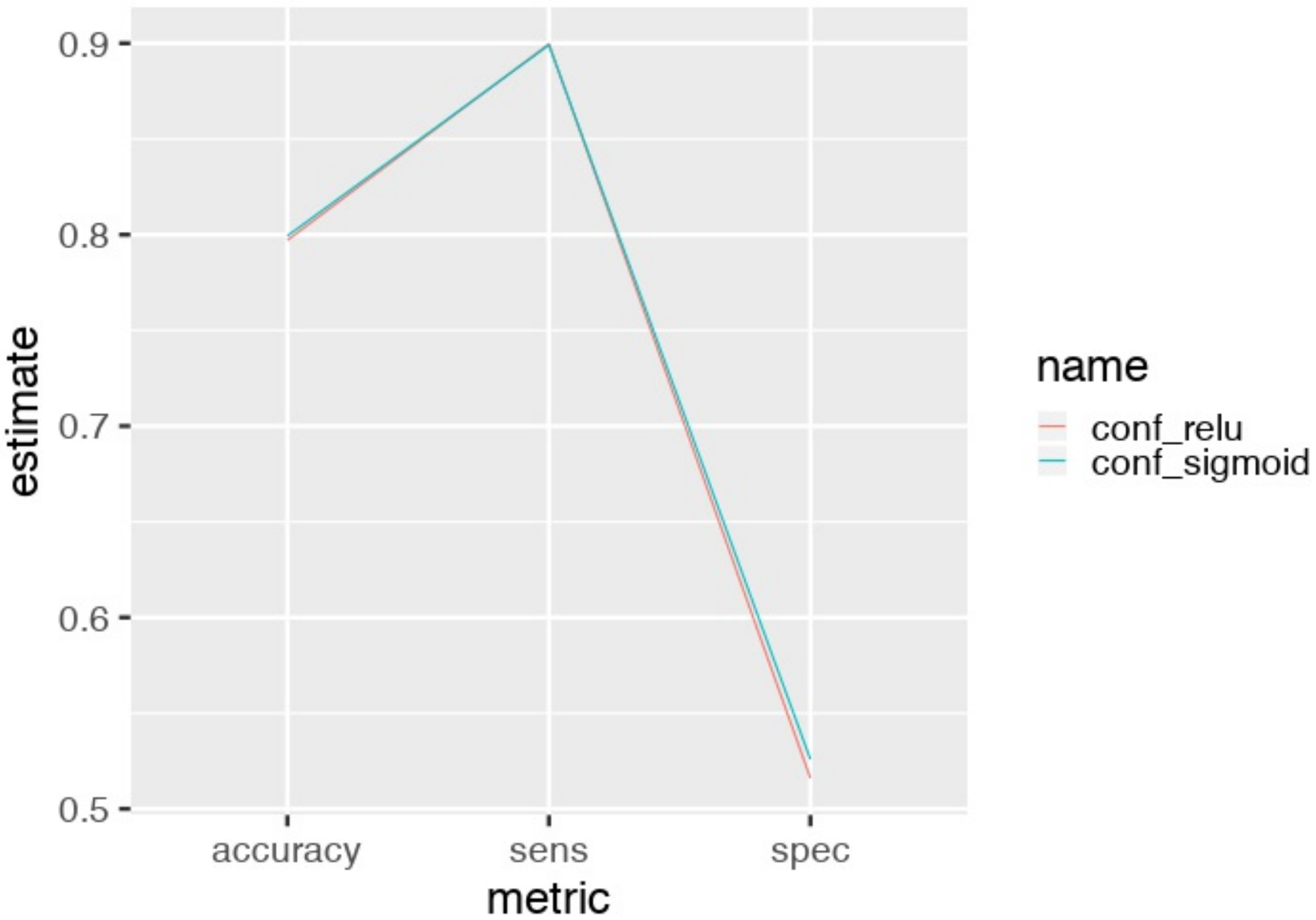
Run the project in make.R.

```
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.

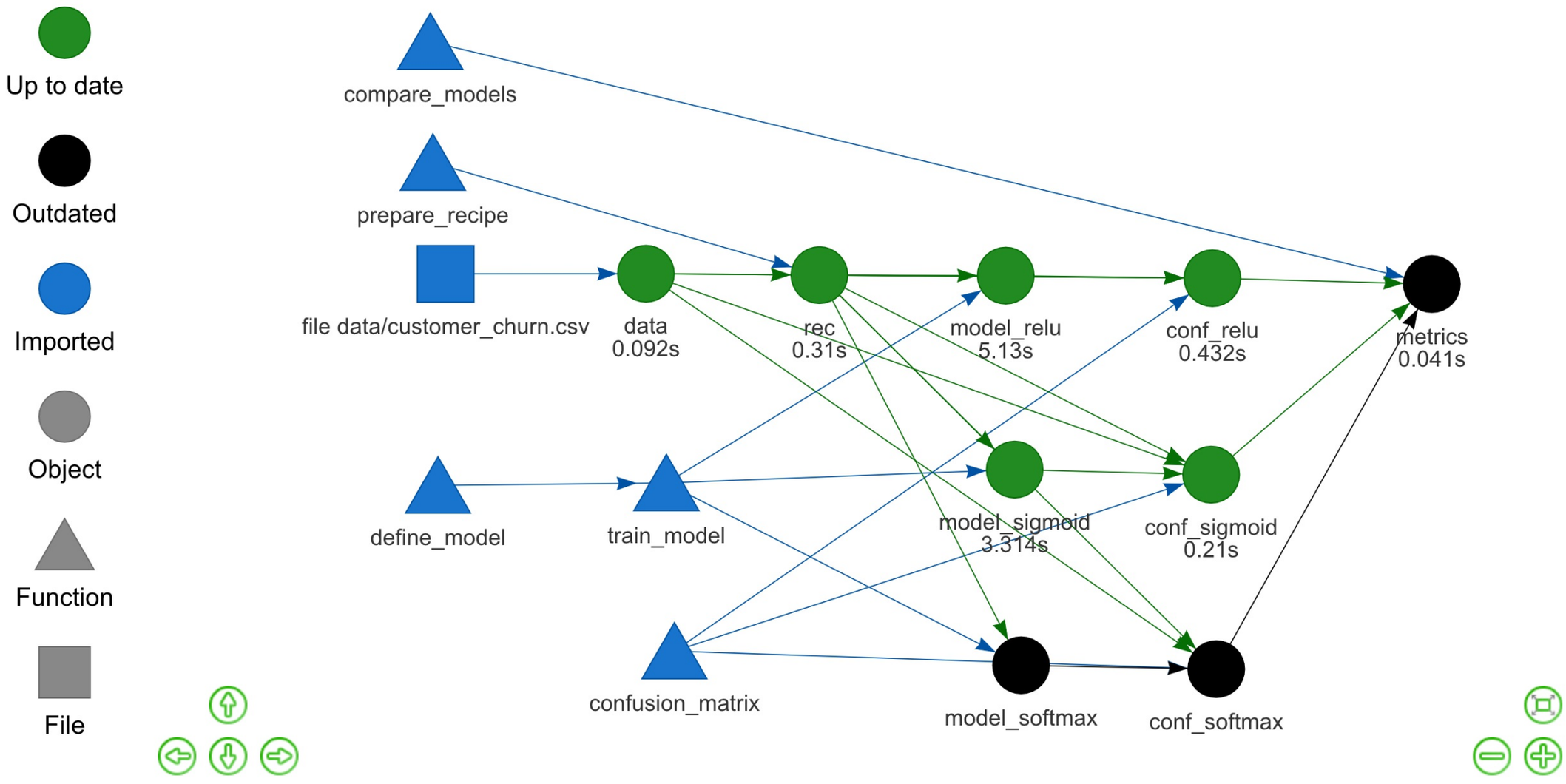
```
readd(metrics) # See also loadd()
```



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()



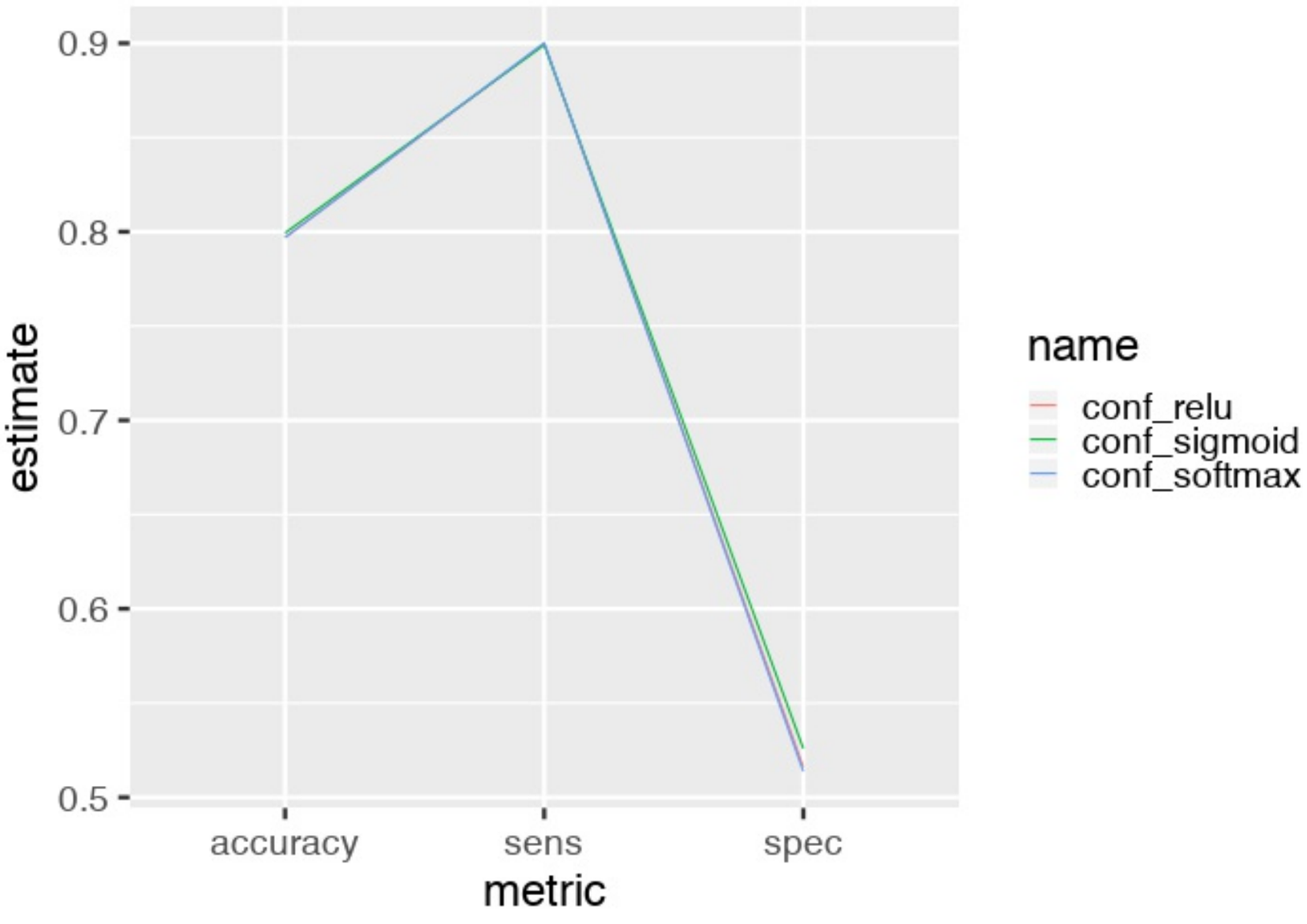
Refresh the results in make.R.

```
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  prop act
##    <chr>   <lgl>   <chr>  <lgl>  <chr> <chr>      <int>   <dbl> <dbl> <ch
##  1 conf_r… TRUE    2019-… TRUE   2734… confusio… 4.05e8  0.232   NA   <NA
##  2 conf_s… TRUE    2019-… TRUE   0393… confusio… 1.93e9  0.203   NA   <NA
##  3 conf_s… TRUE    2019-… TRUE   c5e6… confusio… 1.80e9  0.428   NA   <NA
##  4 data    TRUE    2019-… TRUE   ca84… "read_cs… 1.29e9  0.039    0.3 <NA
##  5 metrics FALSE   2019-… TRUE   5c09… compare_… 1.21e9  0.0230  NA   <NA
##  6 metrics TRUE    2019-… TRUE   4c59… compare_… 1.21e9  0.021   NA   <NA
##  7 model_… TRUE    2019-… TRUE   23be… "train_m… 1.47e9  4.92    NA   relu
##  8 model_… TRUE    2019-… TRUE   0eaa… "train_m… 1.26e9  3.26    NA   sigm
##  9 model_… TRUE    2019-… TRUE   52d6… "train_m… 8.05e8  3.61    NA   soft
## 10 rec     TRUE    2019-… TRUE   40e5… prepare_… 6.29e8  0.168   NA   <NA
```

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
##   0.095   0.038   0.272
```

- 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")
```

- Workshop materials:

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

- Example code from these slides:

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

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