Reproducible computation at scale in R

Will Landau

Copyright Eli Lilly and Company

Large statistical computation

- Bayesian data analysis
- Bayesian network meta-analysis
- Graph-based multiple comparison procedures
- Subgroup identification
- Predictive modeling
- Deep neural networks
- PK/PD modeling
- Clinical trial simulation
- Target identification

Copyright Eli Lilly and Company

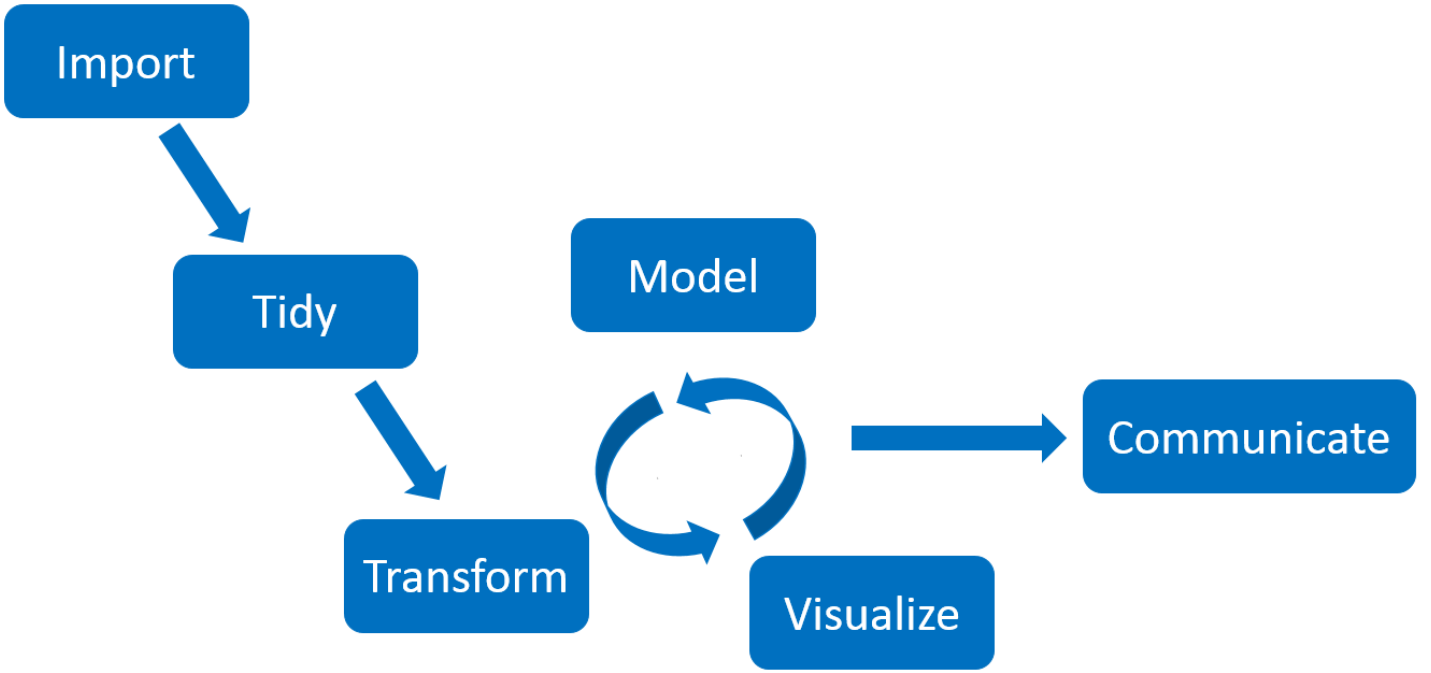
Common features

1. Heavy use of the R language.
2. Long runtimes.
3. Multiple sub-tasks.
4. Frequent changes to code and data.

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

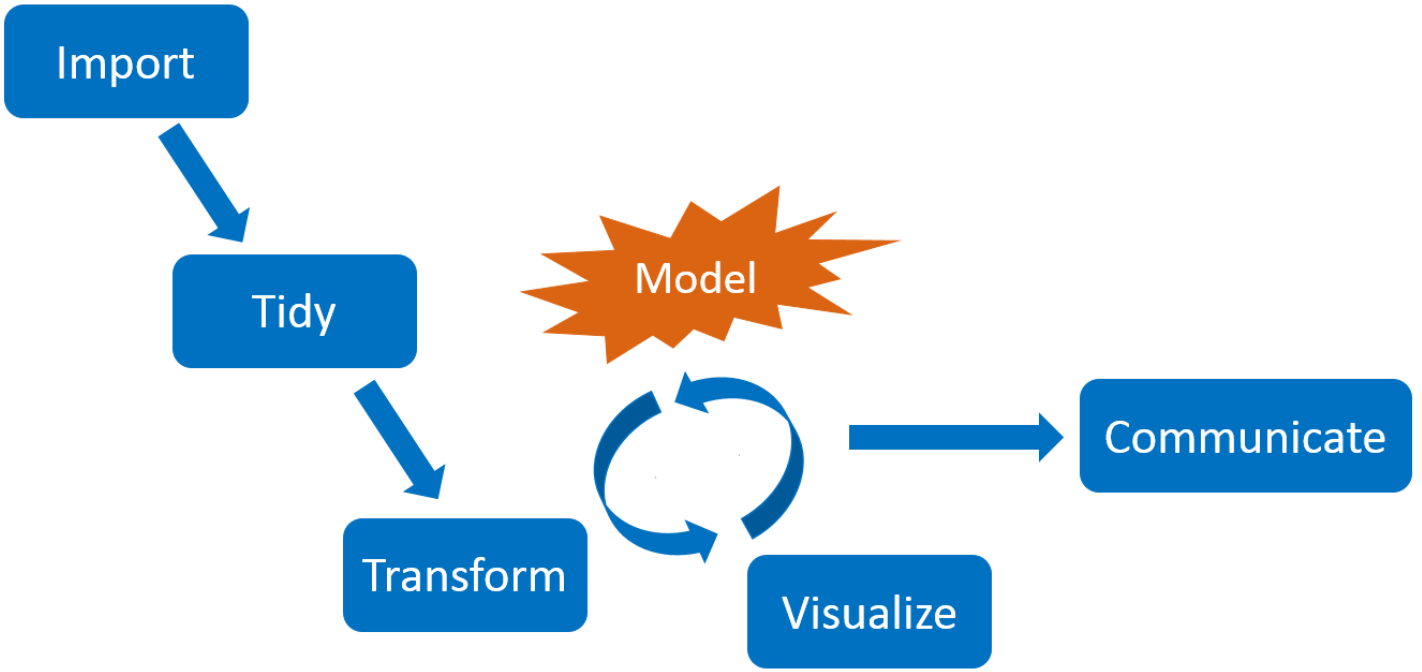
Copyright Eli Lilly and Company

Interconnected tasks



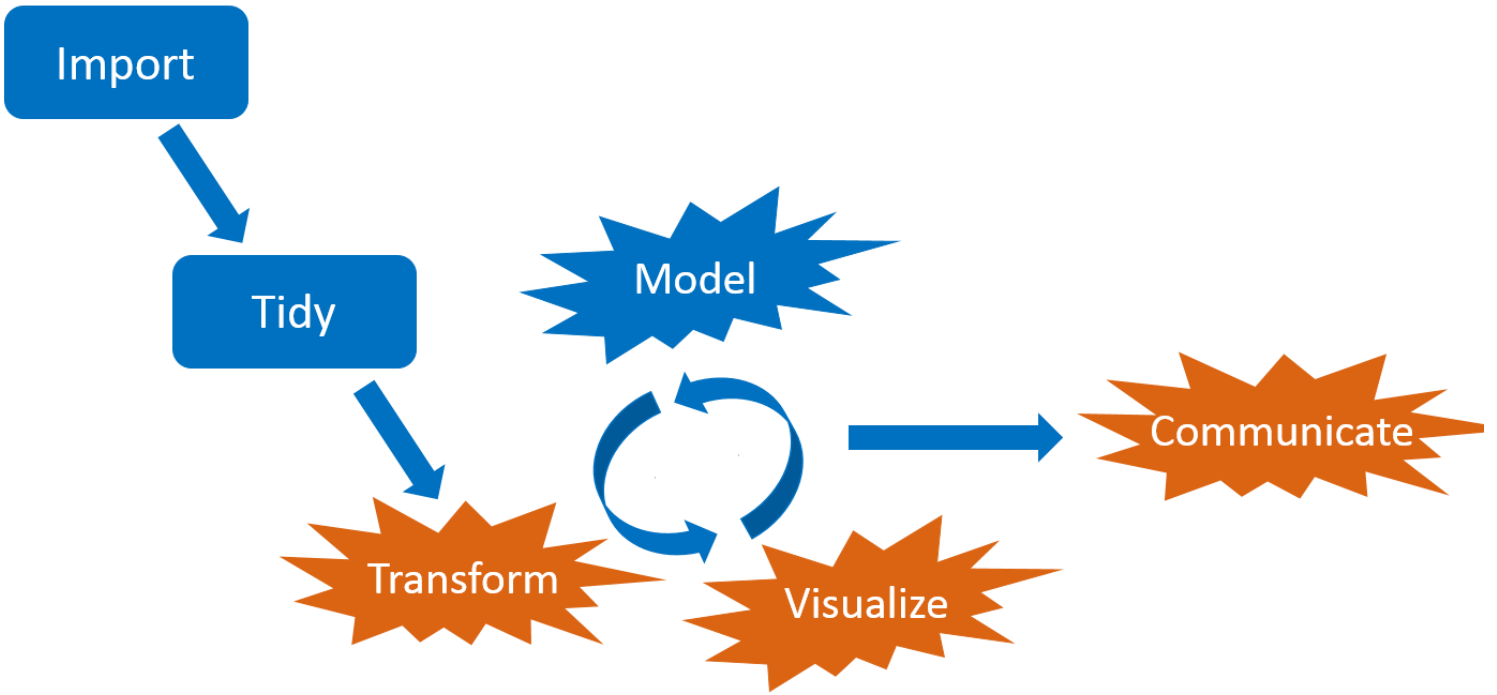
Copyright Eli Lilly and Company

Changes



Copyright Eli Lilly and Company

Consequences

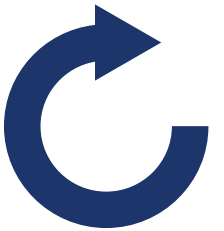


Copyright Eli Lilly and Company

Pipeline tools and workflow managers

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.

Copyright Eli Lilly and Company

What distinguishes `targets`?

- Fundamentally designed for R.
- Supports a clean, modular, function-oriented programming style.
- Abstracts files as R objects and automatically manages data.
- Surpasses the permanent limitations of its predecessor, `drake`: https://wlandau.github.io/targets/articles/need.html

Copyright Eli Lilly and Company

What about `drake`?

- `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.
  1. Stronger guardrails by design.
  2. A friendlier, lighter, more transparent data management system.
  3. Show which *functions* are up to date.
  4. More flexible dynamic branching, including compatibility with `dplyr::group_by()`.
  5. Improved parallel efficiency.
  6. Designed for custom user-side metaprogramming and target archetypes: https://wlandau.github.io/tarchetypes/.
- The statement of need describes the details: https://wlandau.github.io/targets/articles/need.html.

Copyright Eli Lilly and Company

Guardrails in `targets`

- The only way to use `targets` is the correct way.
- Main guardrails:
  1. Always run in a fresh R process (unless you deliberately configure `targets` for debugging).
  2. Require a `_targets.R` configuration file in the project root.
  3. Require the `_targets/` data store to always be in the project root.

Copyright Eli Lilly and Company

drake's cache

```
.drake/
├── config/
├── data/
├──── 17bfcef645301416.rds
├──── 21935c86f12692e2.rds
├──── 37caf5df2892cfc4.rds
├──── ...
├── drake/
├── keys/
├──── memoize/
├──── meta/
├──── objects/
├──── ...
└── scratch/
```

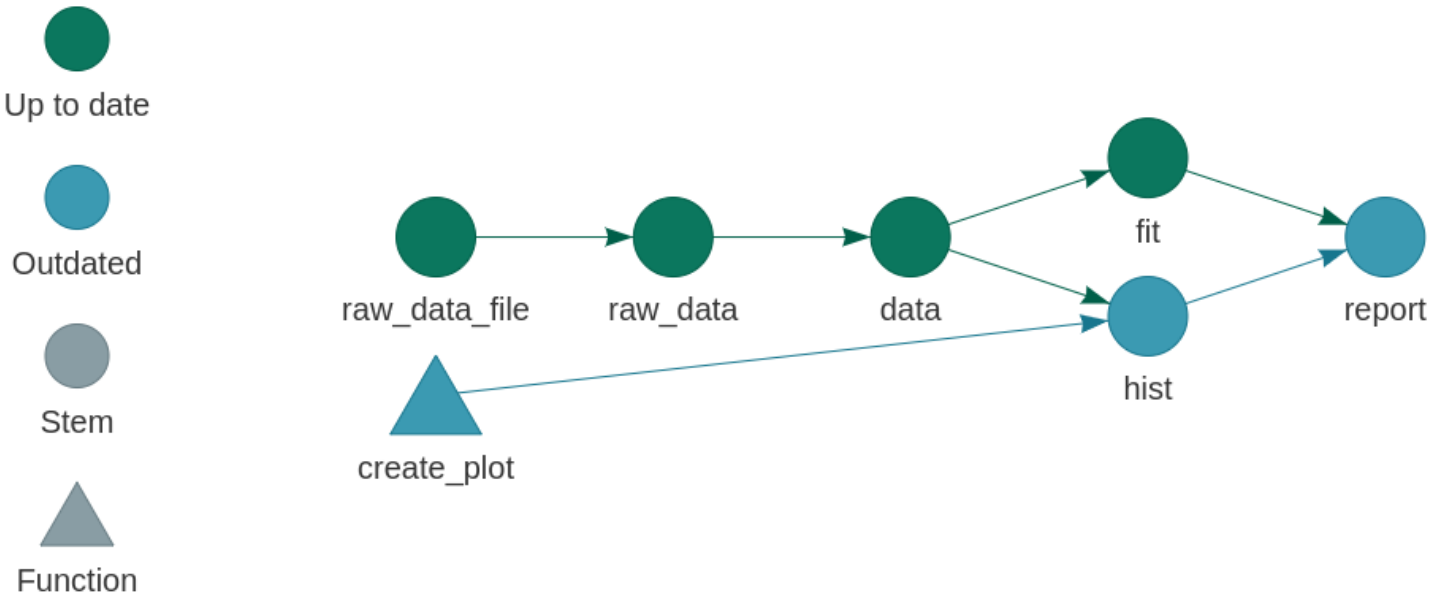
Copyright Eli Lilly and Company

The data store in `targets`

```
_targets/
├── meta/
├────── meta
├────── progress
├── objects/
├────── target_name_1
├────── target_name_2
├────── target_name_3
└────── ...
```

Copyright Eli Lilly and Company

Show which functions are out of date



Copyright Eli Lilly and Company

Dynamic branching with `dplyr`

```
library(dplyr)
library(targets)
data.frame(
  x = seq_len(6),
  id = rep(letters[seq_len(3)], each = 2)
) %>%
  group_by(id) %>%
  tar_group()
#> # A tibble: 6 x 3
#> # Groups:   id [3]
#>       x id    tar_group
#>   <int> <chr>     <int>
#> 1     1 a             1
#> 2     2 a             1
#> 3     3 b             2
#> 4     4 b             2
#> 5     5 c             3
#> 6     6 c             3
```

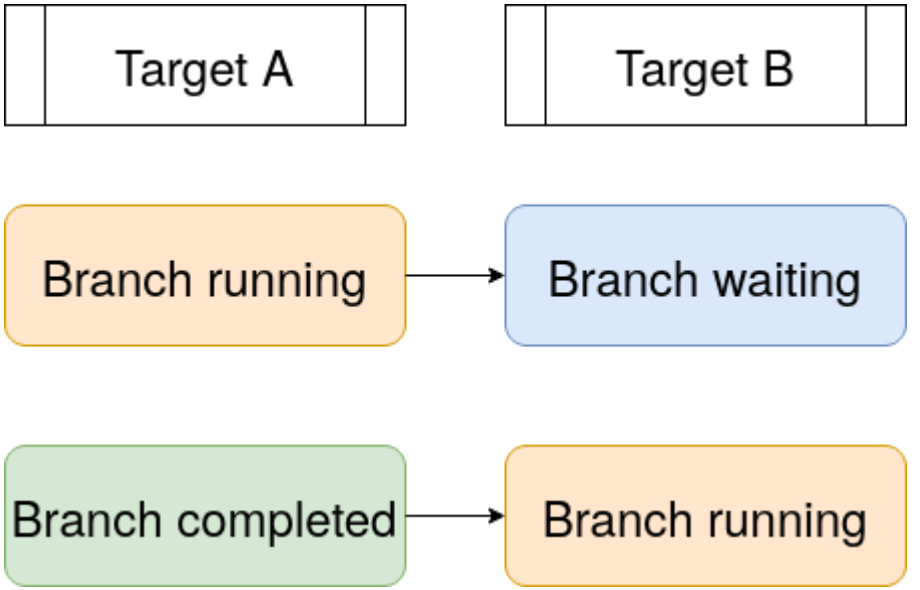
Copyright Eli Lilly and Company

Define a target with groups.

```
tar_target(
  data,
  data.frame(
    x = seq_len(6),
    id = rep(letters[seq_len(3)], each = 2)
  ) %>%
    group_by(id) %>%
    tar_group(),
  iteration = "group"
)
```

Copyright Eli Lilly and Company

Parallel efficient dynamic branching



Copyright Eli Lilly and Company

Metaprogramming

- `tar_target_raw()` avoids non-standard evaluation and supports third-party metaprogramming.
- The following are equivalent ways to define a target.

```
# For most users:
tar_target(data, simulate_data(), pattern = map(index))

# For developers who metaprogram reusable pipeline archetypes:
tar_target_raw(
  "data",
  quote(simulate_data()),
  pattern = quote(map(index))
)
```

Copyright Eli Lilly and Company

Target archetypes

- The `tarchetypes` package has helpers for commonly used targets: https://wlandau.github.io/tarchetypes/

Function	Target archetype
`tar_render()`	Render a dependency-aware R Markdown report.
`tar_knit()`	Run a dependency-aware `knitr` report.
`tar_change()`	Always run a target when a custom object changes.
`tar_force()`	Always run a target when a custom condition is true.
`tar_suppress()`	Never run a target when a custom condition is true.
`tar_plan()`	Simplified `drake`-like syntax for `targets` pipelines.

Copyright Eli Lilly and Company

Example targets workflow

- Find the model that best predicts which customers will cancel their telecom subscriptions.
- IBM Watson Telco Customer Churn dataset.
- Workflow principles generalize to the other fields, such as the life sciences.

Copyright Eli Lilly and Company

❌ Move away from numbered imperative scripts.

```
run_everything.R
R/
├── 01-data.R
├── 02-munge.R
├── 03-model.R
├── 04-results.R
└── 05-plot.R
data/
└── customer_churn.csv
```

Copyright Eli Lilly and Company

 Embrace **functions.**

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

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

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

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

Copyright Eli Lilly and Company

Functions for customer churn

```
split_data <- function(churn_file) {
  read_csv(churn_file, col_types = cols()) %>%
    initial_split(prop = 0.3)
}

prepare_recipe <- function(churn_data) {
  churn_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()
}
```

Copyright Eli Lilly and Company

Functions for customer churn

```
define_model <- function(churn_recipe, units1, units2, act1, act2, ac
  # ...
}

train_model <- function(churn_recipe, units1, units2, act1, act2, act
  # ...
}

test_accuracy <- function(churn_data, churn_recipe, churn_model) {
  # ...
}

test_model <- function(churn_data, churn_recipe, units1, units2, acti
  # ...
}

retrain_run <- function(churn_run, churn_recipe) {
  # ...
}
```

Copyright Eli Lilly and Company

Typical project structure

- There are many variations on this theme.

```
_targets.R # Required top-level configuration file.
R/
└── functions.R
data/
└── customer_churn.csv
```

Copyright Eli Lilly and Company

Build up your workflow as a pipeline of targets.

```
# _targets.R
library(targets)
source("R/functions.R")
tar_option_set(packages = c("keras", "tidyverse", "rsample", "recipes
tar_pipeline(
  tar_target(churn_file, "data/customer_churn.csv", format = "file"),
  tar_target(churn_data, split_data(churn_file)),
  tar_target(churn_recipe, prepare_recipe(churn_data))
)
```

Copyright Eli Lilly and Company

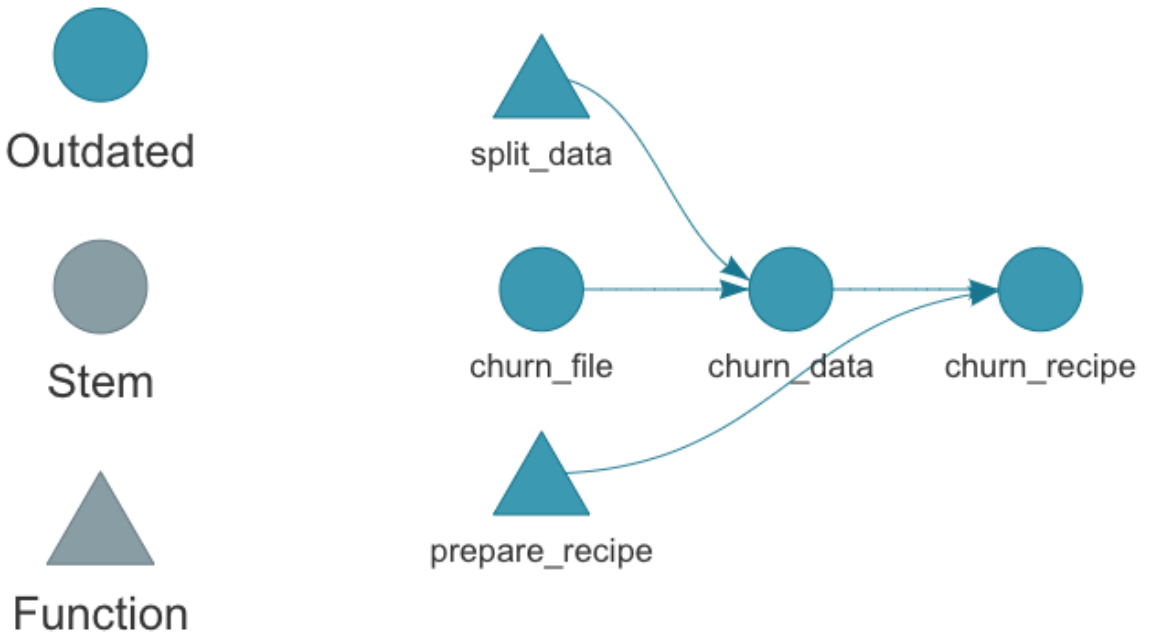
The pipeline is a collection of skippable *targets*.

```
tar_manifest(fields = c("name", "command"))
#> # A tibble: 3 x 2
#>   name         command
#>   <chr>        <chr>
#> 1 churn_recipe "prepare_recipe(churn_data)"
#> 2 churn_data   "split_data(churn_file)"
#> 3 churn_file   "\"data/customer_churn.csv\""
```

Copyright Eli Lilly and Company

targets understands code and data dependencies.

```
tar_visnetwork()
```



Copyright Eli Lilly and Company

Build your first targets.

```
tar_make()
```

```
#> ● run target churn_file
#> ● run target churn_data
#> ● run target churn_recipe
```

Copyright Eli Lilly and Company

Check the targets for problems

- `tar_load()` and `tar_read()` get targets from the `_targets/` data store.

```
ncol(training(tar_read(churn_data)))
#> [1] 21

tar_load(churn_recipe)
ncol(juice(churn_recipe))
#> [1] 36
```

Copyright Eli Lilly and Company

Build up the pipeline *gradually*.

1. Add a couple targets.
2. Run the pipeline with `tar_make()`.
3. Inspect the new targets with `tar_load()` and `tar_read()`.
4. Repeat often. Not very time-consuming because `tar_make()` skips up-to-date targets.

Copyright Eli Lilly and Company

Add some models.

```
# _targets.R
library(targets)
source("R/functions.R")
tar_option_set(packages = c("keras", "tidyverse", "rsample", "recipes
tar_pipeline(
  tar_target(churn_file, "data/customer_churn.csv", format = "file"),
  tar_target(churn_data, split_data(churn_file)),
  tar_target(churn_recipe, prepare_recipe(churn_data)),
  tar_target(run_relu, test_model(act1 = "relu", churn_data, churn_re
  tar_target(run_sigmoid, test_model(act1 = "sigmoid", churn_data, ch
)
```

Copyright Eli Lilly and Company

Previous work is still up to date.

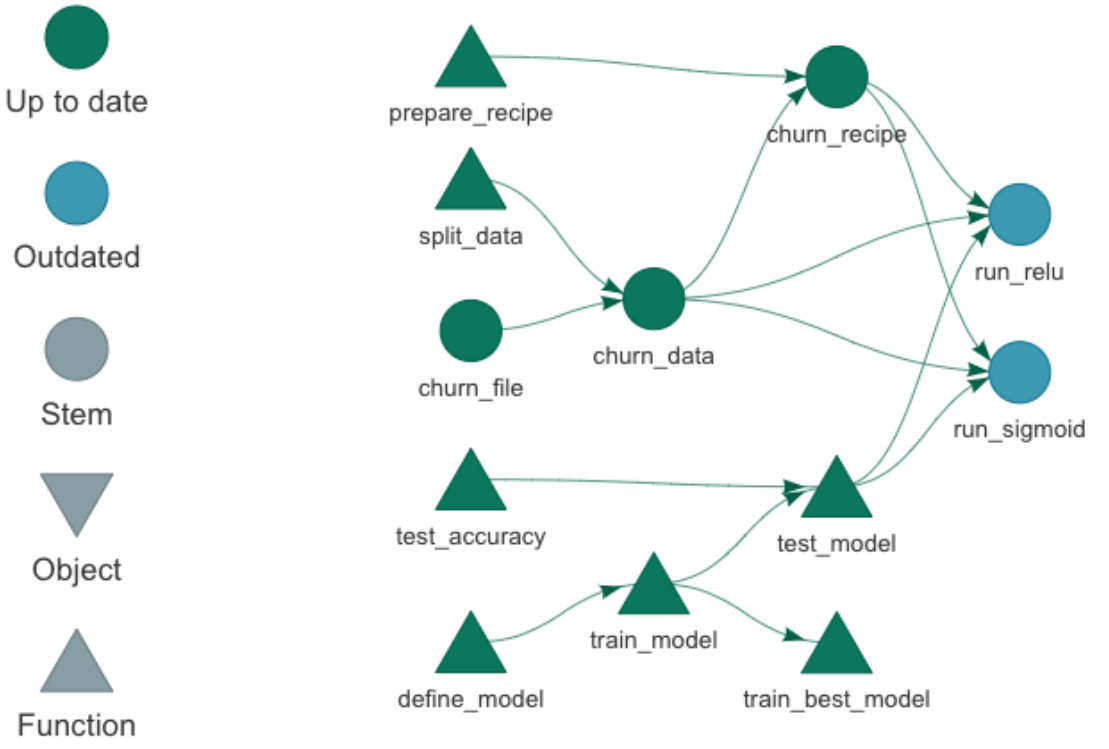
```
tar_outdated()
```

```
#> [1] "run_relu"    "run_sigmoid"
```

Copyright Eli Lilly and Company

Previous work is still up to date.

```
tar_visnetwork()
```



Copyright Eli Lilly and Company

Up-to-date targets are skipped.

```
tar_make()
```

```
#> ✓ skip target churn_file
#> ✓ skip target churn_data
#> ✓ skip target churn_recipe
#> ● run target run_relu
#> ● run target run_sigmoid
```

Copyright Eli Lilly and Company

Inspect the newest targets.

```
tar_read(run_relu)
#> # A tibble: 1 x 6
#>   accuracy units1 units2 act1  act2  act3
#>      <dbl>  <dbl>  <dbl> <chr> <chr> <chr>
#> 1    0.799     16     16 relu  relu  sigmoid

tar_read(run_sigmoid)
#> # A tibble: 1 x 6
#>   accuracy units1 units2 act1    act2  act3
#>      <dbl>  <dbl>  <dbl> <chr>   <chr> <chr>
#> 1    0.799     16     16 sigmoid relu  sigmoid
```

Copyright Eli Lilly and Company

Find the best model

```
# _targets.R
library(targets)
source("R/functions.R")
tar_option_set(packages = c("keras", "tidyverse", "rsample", "recipes
tar_pipeline(
  ...,
  tar_target(run_relu, test_model(act1 = "relu", churn_data, churn_re
  tar_target(run_sigmoid, test_model(act1 = "sigmoid", churn_data, ch
  tar_target(
    best_run,
    bind_rows(run_relu, run_sigmoid) %>%
      top_n(1, accuracy) %>%
      head(1)
  ),
  tar_target(
    best_model,
    retrain_run(best_run, churn_recipe),
    format = "keras"
  )
)
```

Copyright Eli Lilly and Company

Find the best model

```
tar_make()
```

```
#> ✓ skip target churn_file
#> ✓ skip target churn_data
#> ✓ skip target churn_recipe
#> ✓ skip target run_relu
#> ✓ skip target run_sigmoid
#> ● run target best_run
#> ● run target best_model
```

Copyright Eli Lilly and Company

Find the best model

```
tar_read(best_model)
#> Model
#> Model: "sequential_2"
#> _________________________________________________________________
#> Layer (type)                 Output Shape              
#> =================================================================
#> dense_6 (Dense)              (None, 16)                
#> _________________________________________________________________
#> dropout_4 (Dropout)          (None, 16)                
#> _________________________________________________________________
#> dense_7 (Dense)              (None, 16)                
#> _________________________________________________________________
#> dropout_5 (Dropout)          (None, 16)                
#> _________________________________________________________________
#> dense_8 (Dense)              (None, 1)                 
#> =================================================================
#> Total params: 865
#> Trainable params: 865
#> Non-trainable params: 0
#> _________________________________________________________________
```

Copyright Eli Lilly and Company

Try another model.

```
# _targets.R
library(targets)
source("R/functions.R")
tar_option_set(packages = c("keras", "tidyverse", "rsample", "recipes
tar_pipeline(
  ...,
  tar_target(run_relu, test_model(act1 = "relu", churn_data, churn_re
  tar_target(run_sigmoid, test_model(act1 = "sigmoid", churn_data, ch
  tar_target(run_softmax, test_model(act1 = "softmax", churn_data, ch
  tar_target(
    best_run,
    bind_rows(run_relu, run_sigmoid, run_softmax) %>%
      top_n(1, accuracy) %>%
      head(1)
  ),
  ...
)
```

Copyright Eli Lilly and Company

What gets done stays done.

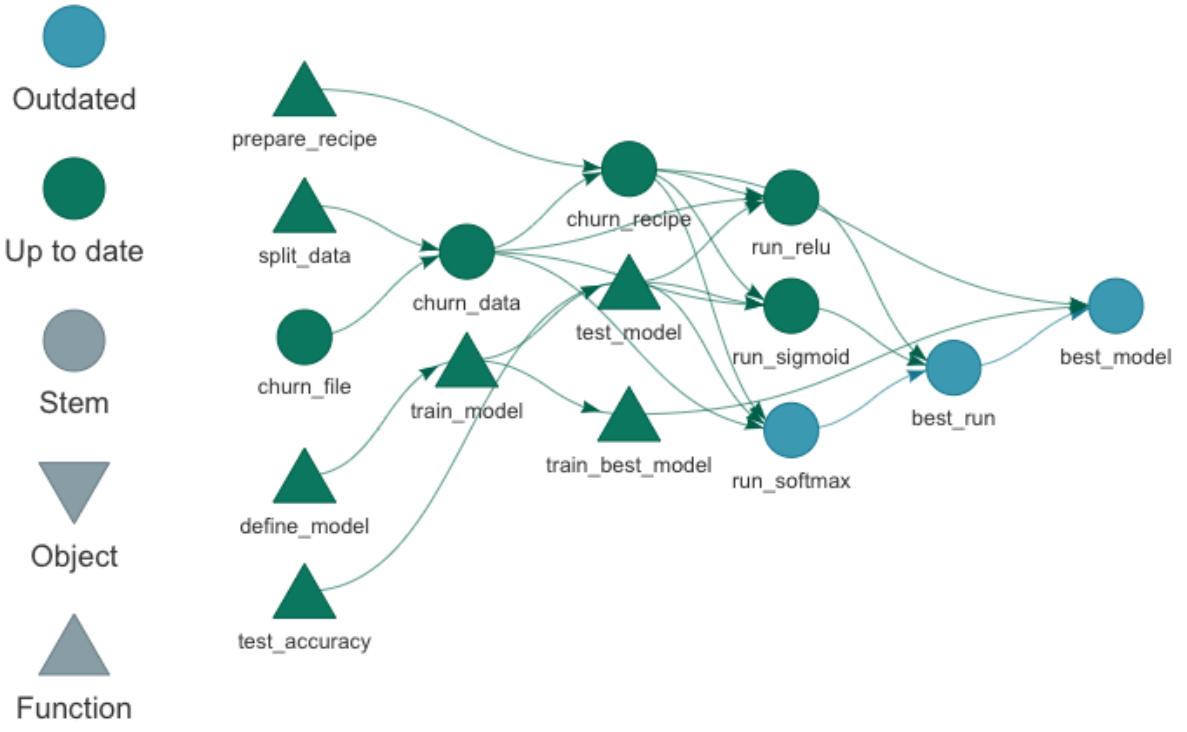
```
tar_outdated()
```

```
#> [1] "run_softmax" "best_run"    "best_model"
```

Copyright Eli Lilly and Company

What gets done stays done.

```
tar_visnetwork()
```



Copyright Eli Lilly and Company

New best model?

- Only if the new run beats the old runs, which would invalidate target `best_run`.
- Otherwise, `drake` does not bother to retrain the best model.

```
tar_make()
#> ✓ skip target churn_file
#> ✓ skip target churn_data
#> ✓ skip target churn_recipe
#> ✓ skip target run_relu
#> ✓ skip target run_sigmoid
#> ● run target run_softmax
#> ● run target best_run
#> ✓ skip target best_model
```

Copyright Eli Lilly and Company

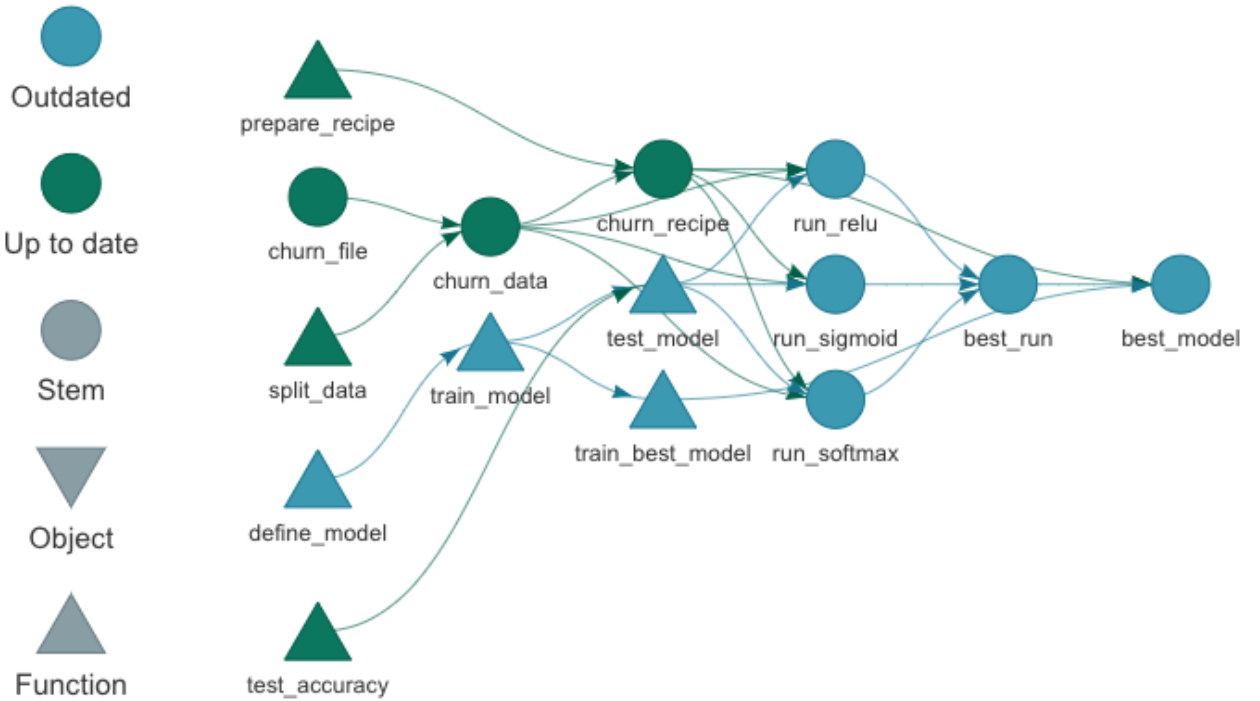
What if we need to change a function?

```
define_model <- function(churn_recipe, units1, units2, act1, act2, ac
  input_shape <- ncol(
    juice(churn_recipe, all_predictors(), composition = "matrix")
  )
  keras_model_sequential() %>%
    layer_dense(
      units = units1,
      kernel_initializer = "uniform",
      activation = act1,
      input_shape = input_shape
    ) %>%
    layer_dropout(rate = 0.2) %>% # previously 0.1
    layer_dense(
      units = units2,
      kernel_initializer = "uniform",
      activation = act2
    ) %>%
    layer_dropout(rate = 0.1) %>%
    ...
```

Copyright Eli Lilly and Company

Show invalidated functions and targets.

```
tar_visnetwork()
```



Copyright Eli Lilly and Company

Rerun invalidated targets.

```
tar_make()
#> ✓ skip target churn_file
#> ✓ skip target churn_data
#> ✓ skip target churn_recipe
#> ● run target run_relu
#> ● run target run_sigmoid
#> ● run target run_softmax
#> ● run target best_run
#> ● run target best_model
```

Copyright Eli Lilly and Company

Similar story if the data file changes.

```
tar_make()
#> ● run target churn_file
#> ● run target churn_data
#> ● run target churn_recipe
#> ● run target run_relu
#> ● run target run_sigmoid
#> ● run target run_softmax
#> ● run target best_run
#> ● run target best_model
```

Copyright Eli Lilly and Company

Evidence of reproducibility

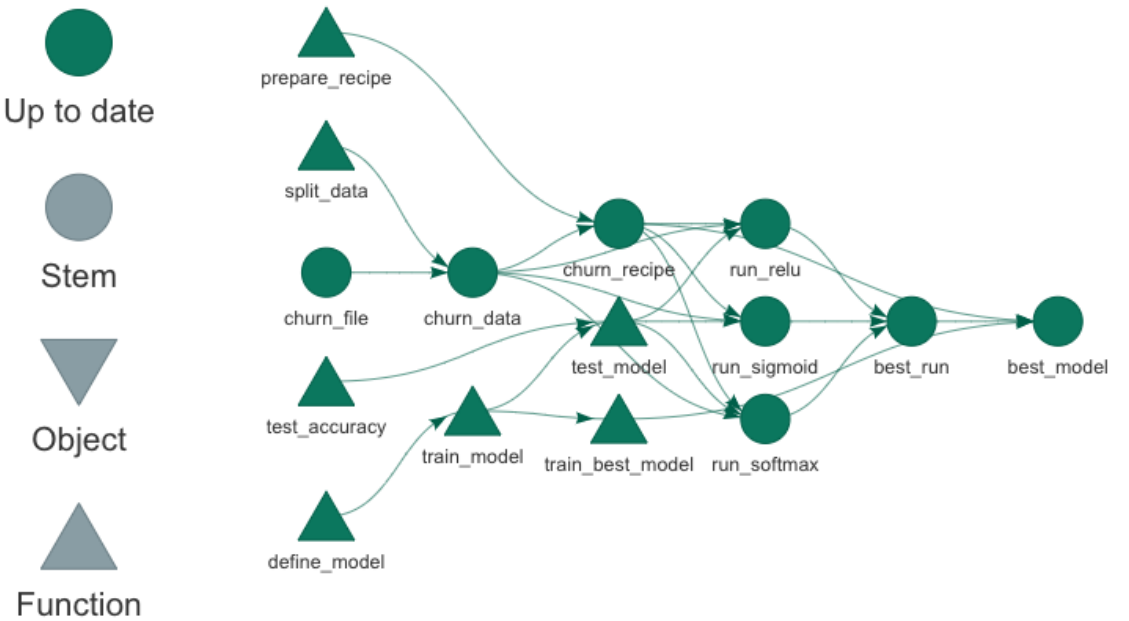
```
tar_make(plan)
#> ✓ skip target churn_file
#> ✓ skip target churn_data
#> ✓ skip target churn_recipe
#> ✓ skip target run_relu
#> ✓ skip target run_sigmoid
#> ✓ skip target run_softmax
#> ✓ skip target best_run
#> ✓ skip target best_model
#> ✓ Already up to date.
```

Copyright Eli Lilly and Company

Evidence of reproducibility

```
tar_outdated()
#> character(0)

tar_visnetwork()
```



Copyright Eli Lilly and Company

Resources

- Get `targets`:

```
install.packages("remotes")
remotes::install_github("wlandau/targets")
```

- Code: https://github.com/wlandau/targets-keras
- RStudio Cloud workspace: https://rstudio.cloud/project/1430828/
- These slides: https://wlandau.github.io/targets-tutorial
- Tutorial materials: https://github.com/wlandau/targets-tutorial
- Development repository: https://github.com/wlandau/targets
- Full user manual: https://wlandau.github.io/targets-manual/
- Reference website: https://wlandau.github.io/targets/

Copyright Eli Lilly and Company

Thanks

- Edgar Ruiz
- example code

- Matt Dancho
- blog post

Copyright Eli Lilly and Company

The tutorial

1. Sign up for a free account at https://rstudio.cloud.
2. Log into https://rstudio.cloud/project/1512447.
3. Work through the R notebooks in order.
4. Optional: revisit the material again later at https://github.com/wlandau/targets-tutorial.

Topic	Notebook
Functions	`1-functions.Rmd`
Pipelines	`2-pipelines.Rmd`
Changes	`3-changes.Rmd`
Files	`4-files.Rmd`
Branching	`5-branching.Rmd`
Debugging	`6-debugging.Rmd`
Challenge	`7-challenge.Rmd`

Copyright Eli Lilly and Company