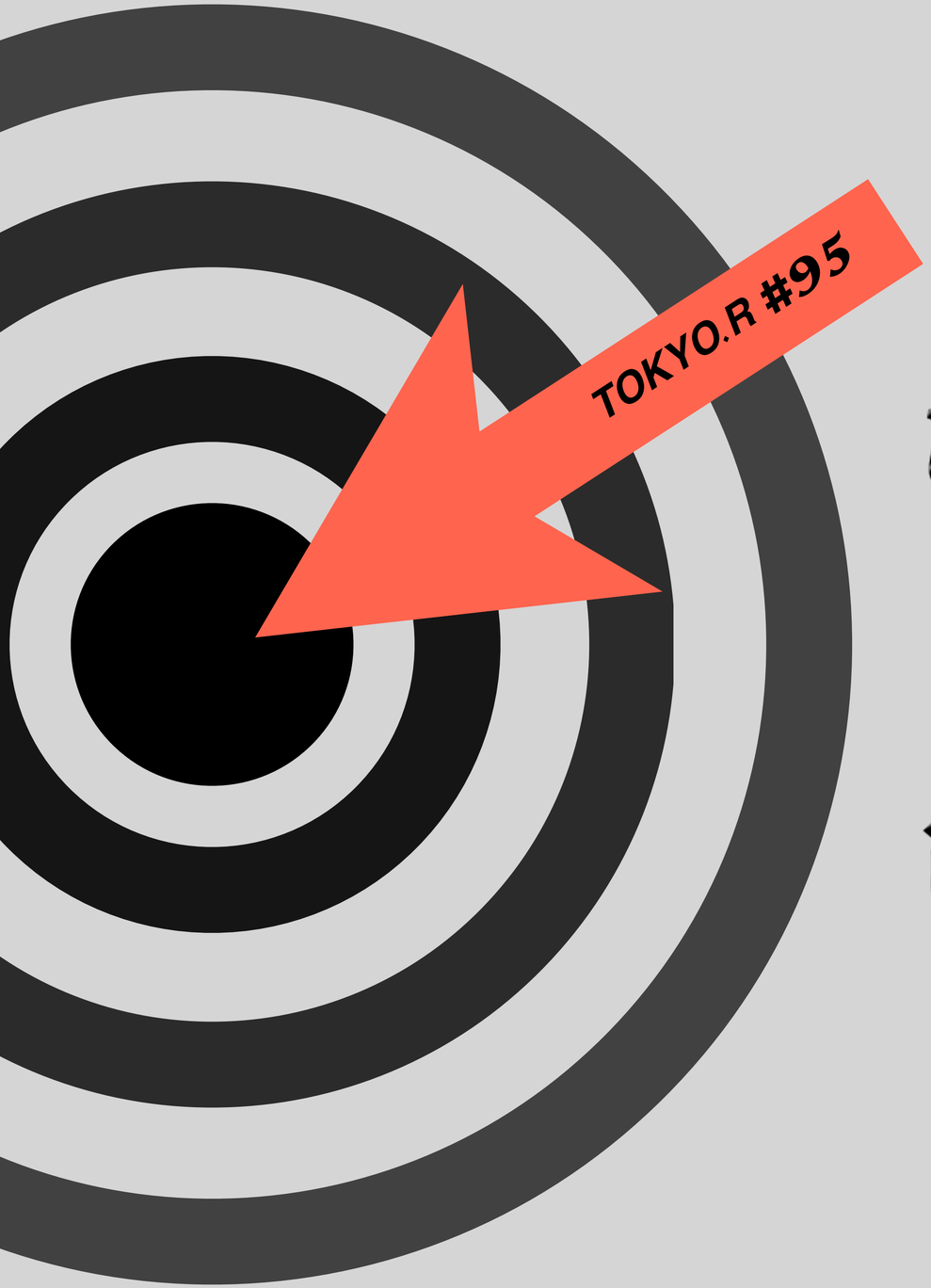
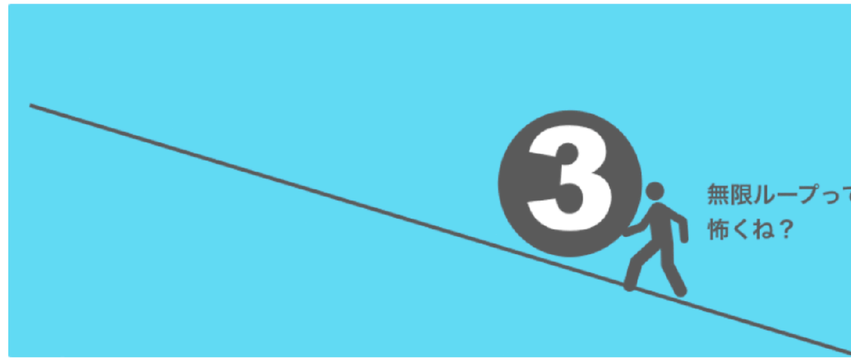


{targets}で
ワークフローを
管理せよ

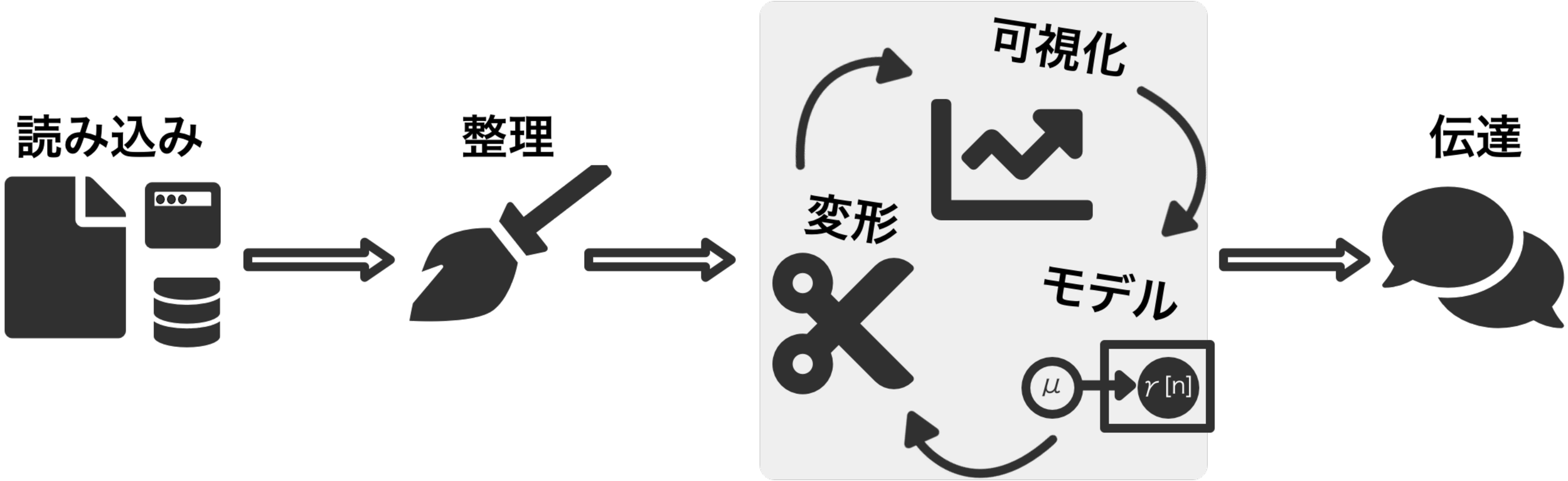
Shinya Uryu
@u_ribo

シーシュポスの岩

作業のやり直しに苦しむデータ分析者たち

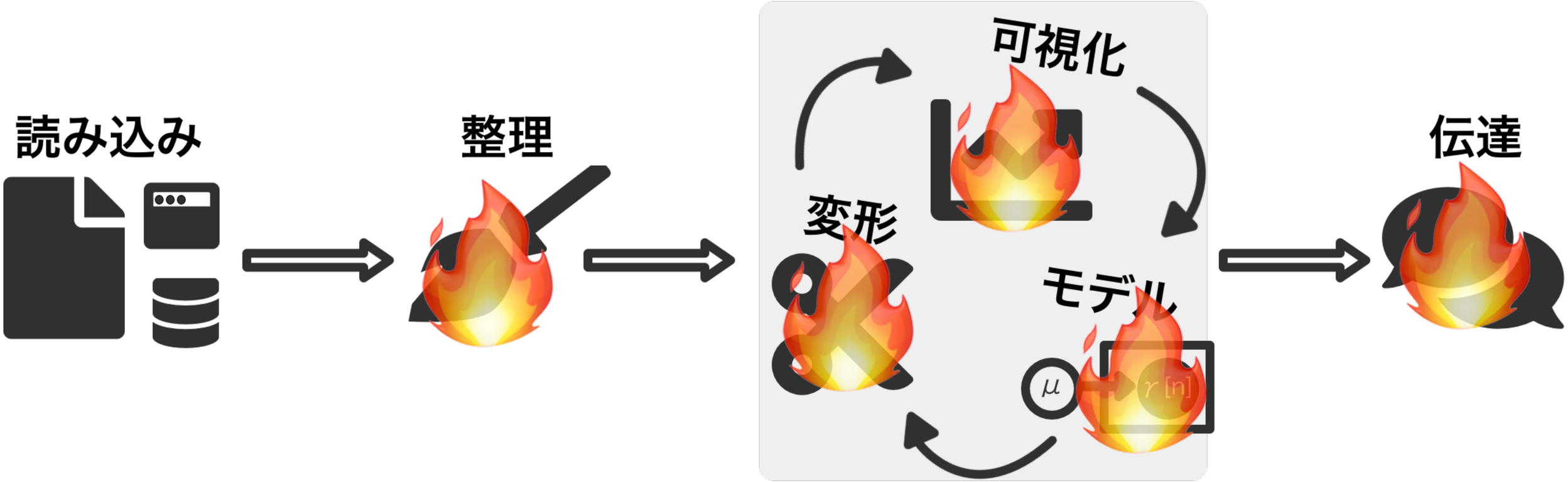


ワークフローの依存関係



Garrett and Hadley (2016) より作成

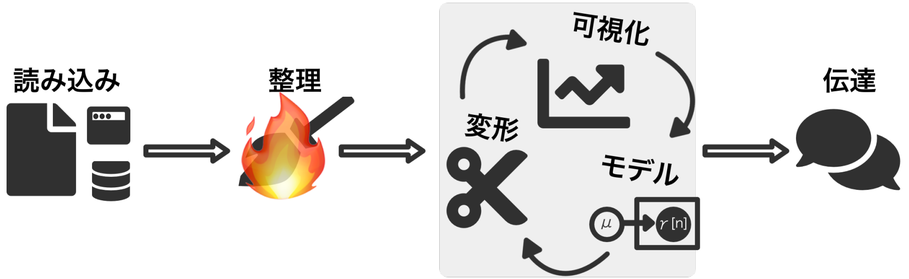
ワークフローの依存関係



上流で行った変更が下流に影響

再現性と実行時間のジレンマ

ワークフローに変更が加わった時
どうする？



	再現性	実行時間	フォルダ管理
コードを残す	○	×	手間がかかる
結果（データ）を残す	×	○	手間がかかる

{targets}

Dynamic Function-Oriented 'Make'-Like Declarative Workflows

- 統計・データサイエンスのためのパイプラインツール
- 関数指向プログラミングと宣言型のワークフロー が特徴
- コードとデータの管理が容易
- {drake}パッケージの後継

*{targets}*を勧める理由

- コード、データ管理の煩わしさからの脱却
- 修正・追加作業の際の負担が減少
  ワークフローによる依存関係の把握
- 作業の流れの視覚化
  データ・コードのブラックボックス化を防ぐ
- 全体の処理を簡潔に
  処理を関数化することで保守性も向上

今日話（せ|さ）ないこと

- Target Markdown　R Markdown 文書内部でパイプラインを定義
  Target Markdown 検証メモ - terashim.com
  https://terashim.com/posts/target-markdown/
- デバッグの方法
- _targets/オブジェクトの管理　など

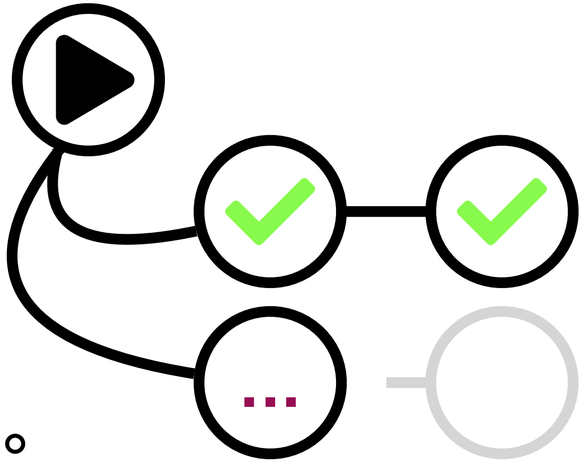
これらの内容はマニュアルを参考
https://books.ropensci.org/targets/

*{targets}*のパイプライン

ワークフロー全体の依存関係
（コード、データ、上流での処理内容）を監視

実行結果を内部に保存。再実行時、前回の実行時から
変更のない処理（ターゲット）をスキップ

→ 実行時間の短縮、
再現性が保証される。



フォルダ管理の煩わしさから解放される。
Stanなどの実行に時間を要する処理や機械学習の実験に適する。

{*targets*}の基礎

サンプルリポジトリ

https://github.com/uribo/talk_211030_tokyor95

{targets}を使ったプロジェクトの基本

```
├─ _targets.R
├─ R/
├── function.R
├─ data/
├── raw_data.csv
└─ hoge.Rproj
```

Targetスクリプトファイル（_targets.R）

{targets}によるパイプラインの
実行手順を記載、
実行に必要なパッケージ・関数読み込みやその他のオプションを含める。

_targets.Rにターゲットとなる処理を記述

3つの処理 ❶❷❸
からなるワークフロー

```
library(targets)
list(
❶ tar_target(
    my_mtcars,
    subset(mtcars, cyl == 6)),
❷ tar_target(
    dist_histogram,
    hist(my_mtcars$disp)),
❸ tar_target(
    lm_res,
    lm(mpg ~ wt, mtcars))
)
```

_targets.R

list()を使ってパイプラインを定義

個別の処理はtar_target()で指定

tar_target(名前, 処理内容)

tar_make()でパイプラインの実行

実行環境とは異なるRセッションで

```
my_mtcars <- subset(mtcars, cyl == 6)
```

が行われるイメージ

tar_make(): パイプラインの実行

最初のtar_make()

```
> library(targets)
> tar_make()
• start target my_mtcars
• built target my_mtcars
• start target lm_res
• built target lm_res
• start target dist_histogram
• built target dist_histogram
• end pipeline
```

二度目のtar_make()

```
> tar_make()
✓ skip target my_mtcars
✓ skip target lm_res
✓ skip target dist_histogram
✓ skip pipeline

>
```

Targetの処理が実行

start → build → データストレージ（後述）に

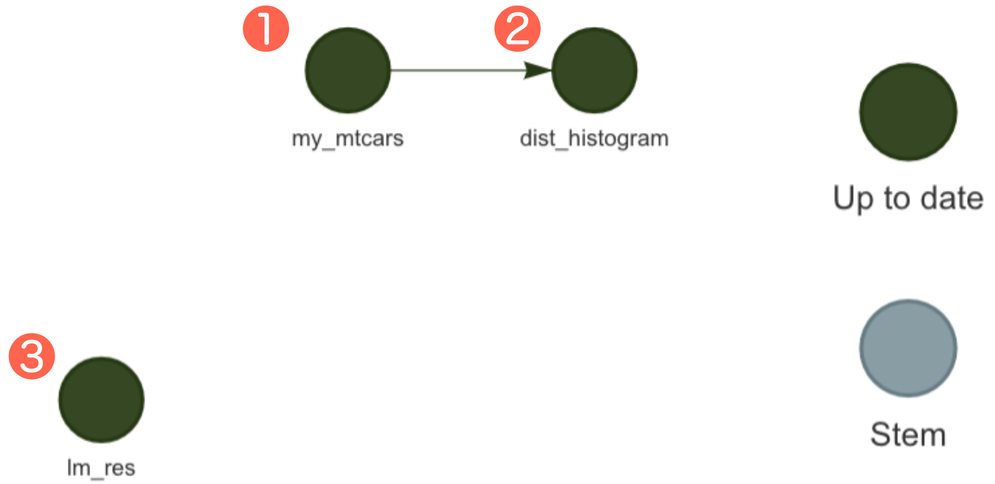
Targetに変更があれば → なければ skip
（キャッシュが利用される）

tar_visnetwork(): ワークフローの可視化

```
library(targets)
list(
  tar_target(
    my_mtcars,
    subset(mtcars, cyl == 6)),
  tar_target(
    dist_histogram,
    hist(my_mtcars$disp)),
  tar_target(
    lm_res,
    lm(mpg ~ wt, mtcars))
)
```

_targets.R

処理内容の依存関係が明確に



どの処理が変更された？

エラー箇所の特定が容易に

tar_read() / tar_load(): 結果の復元

処理結果は _targets/ に蓄積されている

```
tar_read(lm_res)
#> Call:
#>   lm(formula = mpg ~ wt, data = mtcars)
#>
#> Coefficients:
#>   (Intercept)            wt
#> 37.285       -5.344
```

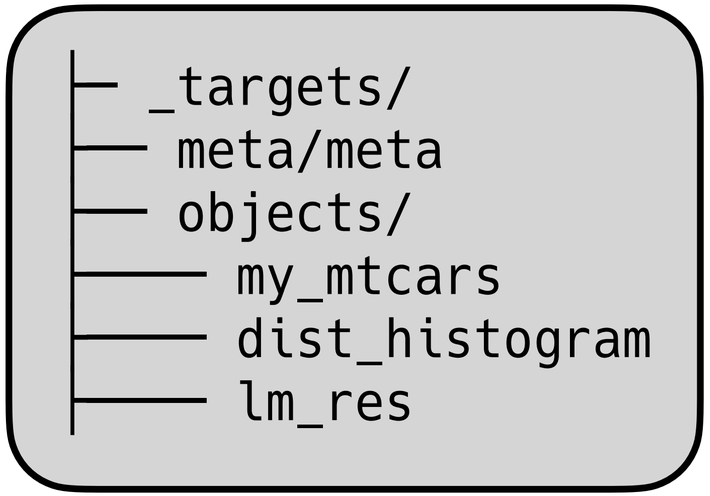
tar_read()で読み込み（出力）

tar_load()で作業空間にオブジェクトとして保存

再現性の向上・時間の短縮

多様なデータストレージ

パイプラインで生成された結果を_targets/ 内で管理



どのようなデータがあるか…

tar_meta()

デフォルトではrds

tar_target(…, format =)

で指定可能

fst　　feather, parquet　　aws

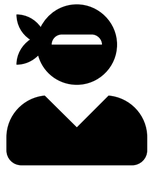
{*targets*}の推し機能

推し1: ブランチ

結果を管理するのは面倒…

モデルのパラメータを変更して結果を比較したい
実験として適当な値を入れて試したい

コードに残す？



作図の見た目を変えるのに一部の色だけ変更してみたい

活性化関数の指定を2パターン用意

```
list(…
  tar_target(activations, c("relu",
"sigmoid")),
  tar_target(
    run,
    test_model(act1 = activations,
churn_data, churn_recipe),
    pattern = map(activations)
  ),…)
```

_targets.R

Targetsがよしなに管理！

```
tar_read(…, branches = 2)
```

pattern引数

iteration引数

推し2: 高性能計算

負担の大きな処理も高速に

{clustermq} tar_make_clustermq()　　***{future}*** tar_make_future()

```
options(
  clustermq.scheduler =
"multiprocess")
```

```
library(future)
library(future.callr)
plan(callr)
```

```
list(
  tar_target(data, get_data()),
  tar_target(fast_fit, fit_small_model(data)),
  tar_target(plot_1, make_plot(fast_fit)))
```

_targets.R

tar_target()内での切り替えは deployment 引数で指定。“main” または “worker”

R targetopia

汎用的な
処理内容をパッケージ化
用途に特化した関数を提供

`tarchetypes::tar_render()`

Rmdファイルの出力を実行

`stantargets::tar_stan_compile()`

stanファイルのコンパイル