Establishing Pipelines With Targets

Reproducible and Efficient Workflows in R

By Chloe Fung, Hanna Getachew, Laia Domenech Burin

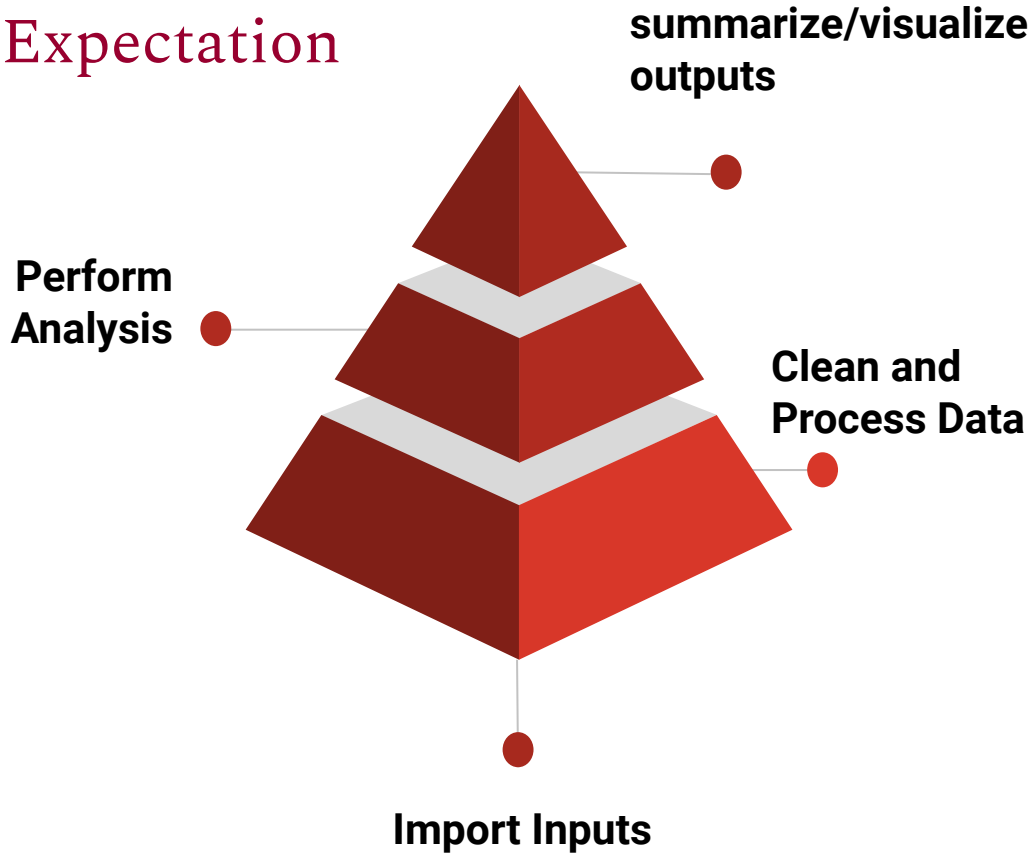
2024

Workshop

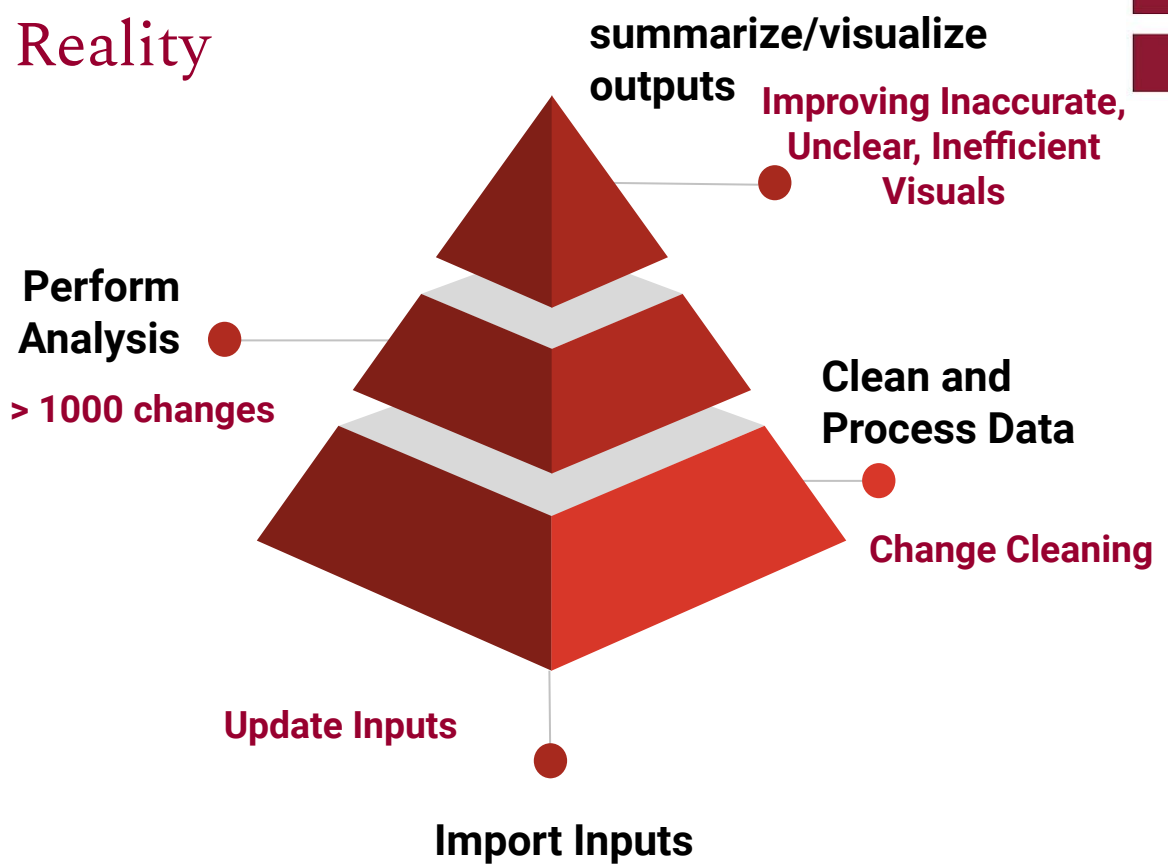
Introduction to Data Science

The Problem:

Expectation



Reality



Challenges in Workflow

- Re-running entire analyses when only small changes occur
- Inefficient use of resources with redundant computations
- Difficulty ensuring reproducibility across multiple runs of code
- Loss of time and trust when working with large datasets

Solution: Introducing targets

- **targets** is an R package designed to optimize and manage data analysis pipelines
- automates tracking of dependencies between steps, ensuring **only necessary parts of the workflow are re-run**
- guarantees reproducibility: Results will always reflect the current state of the code and data

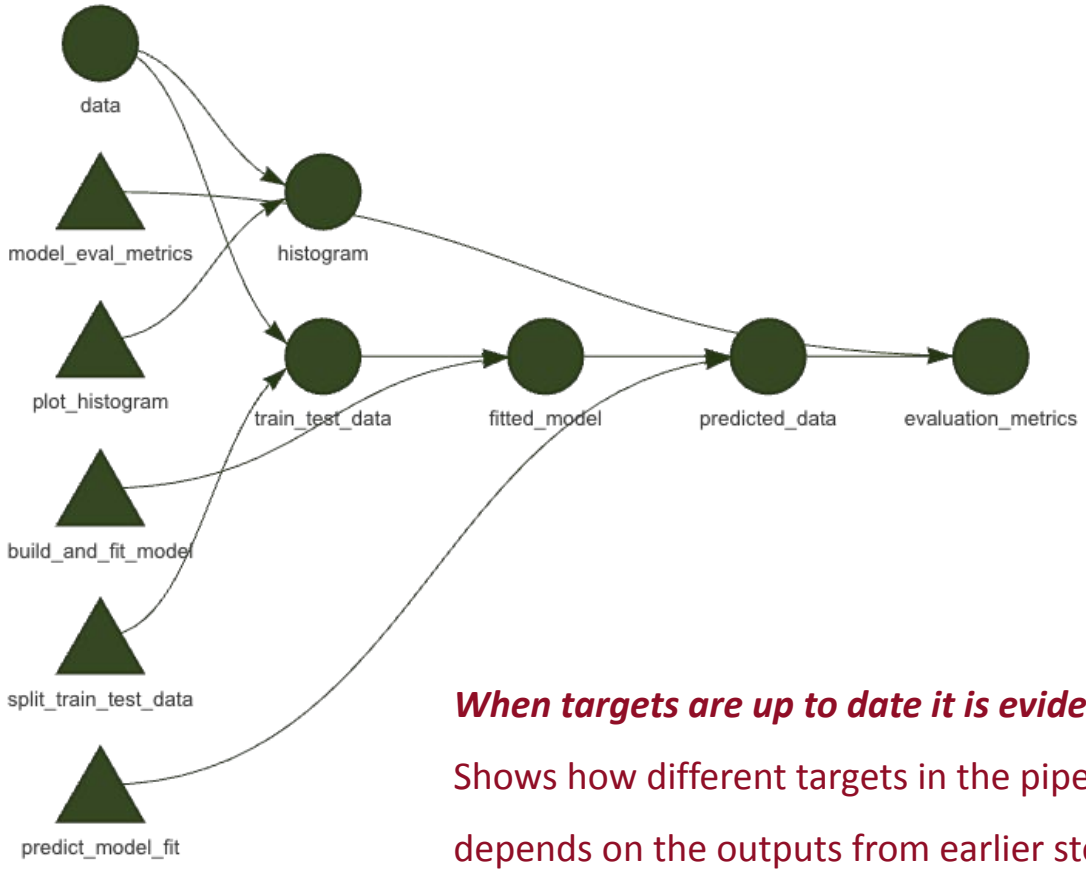
Breaking Down the targets Framework [target]

- Target: is **a single step or task** in your workflow. It represents a discrete operation such as loading data, processing data, or creating a plot.
  - Example:
    - Target 1: Read in the data from a CSV file.
    - Target 2: Filter the data for a specific condition (e.g., filter data for a specific species in the iris dataset).
  - In targets, each of these steps is **defined using tar_target().** By breaking the workflow into separate targets, we can re-run only the necessary steps when something changes, which improves efficiency.

Breaking Down the targets Framework [dependency]

- **Dependency:** occurs when one target relies on the result of another target. In a workflow, certain steps can't run unless earlier steps have been completed. For example, you can't plot data until it has been cleaned and processed. targets **automatically detects these dependencies** and ensures that steps are run in the correct order.
  - Example:
    - Target 1 (raw_data): Loads the raw data (e.g., the iris dataset).
    - Target 2 (processed_data): Filters the raw data. This step depends on the raw data.
    - Target 3 (plot): Creates a plot from the processed data. This step depends on the result of Target 2.
  - In this example, if you change how the data is filtered, only Target 2 and Target 3 will be re-run, not the entire pipeline. targets automatically tracks these dependencies and avoids unnecessary re-computation.

Visualising the dependencies - `using tar_visnetwork()`



When targets are up to date it is evidence that the outputs match the code and the inputs.

Shows how different targets in the pipeline **rely on one another**. Each subsequent target depends on the outputs from earlier steps:

- the split data feeds into the model building process, which in turn informs the prediction (predict_model_fit) and evaluation metrics (model_eval_metrics).
- the histogram target operates independently without interacting with the model related targets.

Breaking Down the targets Framework [pipeline]

- Pipeline: is the entire sequence of connected targets, representing the full workflow of your data analysis. It includes all the steps from data loading, processing, analysis, and visualization, with each target connected through dependencies.
  - Example:
    - Pipeline: The pipeline for analyzing the iris dataset might include the following targets:
    - Load data → Filter data → Summarize data → Create plot

  The pipeline connects these targets, ensuring they execute in the correct order based on dependencies.

Breaking Down the targets Framework [modularity]

- Modularity: means that each target is encapsulated in its own function. This makes your code clean, reusable, and easier to debug or extend. When each target represents a separate function, it becomes easier to isolate issues, test smaller pieces of the workflow, and reuse code in different contexts.
  - Example:
    - Function for loading data: load_data()
    - Function for filtering data: filter_data()
    - Function for summarizing data: summarize_data()
    - Function for creating a plot: create_plot()

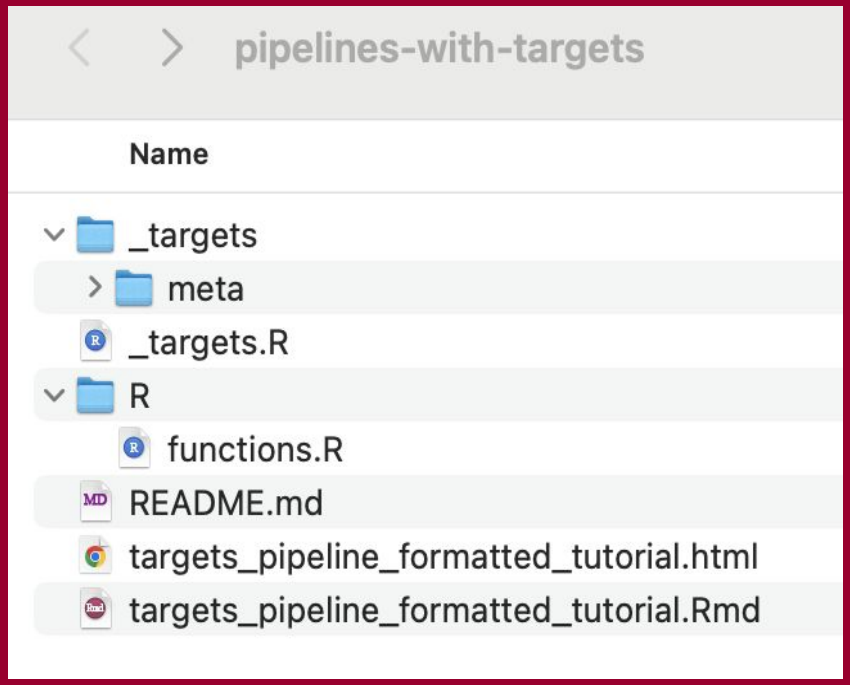
Each function performs **one specific task**, and **each target refers to a function.** Modularity allows you to change one part of the pipeline (e.g., how data is filtered) without affecting the rest of the code.

A Typical targets Pipeline

1. **Write Functions:** Write reusable R functions for each task in your analysis
2. **Define Targets:** Use tar_target() to define each task and its dependencies
3. **Configure Pipeline:** Set global options for your pipeline in _targets.R
4. **Run Pipeline:** Use tar_make() to execute the pipeline
5. **Access Results:** Use tar_read() to read and inspect specific results

Project Structure for targets

- **_targets/:** Automatically generated folder for storing metadata and intermediate results.
- **_targets.R:** Main pipeline configuration file.
- **R/functions.R**: Store custom reusable functions.
    - Stored within the “R” folder
- **README.md:** Documentation file for project instructions.



This structure helps keep the analysis organized, efficient, and easy to maintain.

NOTE: Follow the exact file names and folder structures, otherwise the targets package will not work.

Project Structure: _targets/

- **Purpose:** The _targets/ folder is created by the targets package to store the metadata, progress, and cached results of your pipeline.
- **Subfolder meta/:** Tracks the status of each target (whether it's up to date, needs to be rerun, etc.).

Why it's important:

- **Caching:** This folder ensures that results of completed targets are cached, **so they don't have to be recomputed if nothing has changed.**
- **Tracking Dependencies:** It stores the relationships between different targets, allowing targets to determine which steps need to be re-executed when something in the workflow changes.

Project Structure: _targets.R File

- **Purpose:** The _targets.R file is where we **define our entire analysis pipeline.** It lists all the steps (targets) and their dependencies using tar_target().

Why it's important:

- This file controls the flow of the entire pipeline, ensuring that each step is executed in the correct order and only when necessary.

```r
# Load packages required to define the pipeline:
library(targets)
library(tidyverse)

# Set target options:
tar_option_set(
  packages = c("tibble") # Packages that your targets need for their tasks.
)

# Run the R scripts in the R/ folder with your custom functions:
tar_source()

# Replace the target list below with your own:
list(
  tar_target(
    name = data,
    command = iris
  ),
  tar_target(
    name = summary,
    command = summary_statistics(data)
  ),
  tar_target(
    name = histogram,
    command = plot_histogram(data))
  )
```

Project Structure: R/functions.R

- **Purpose:** Store all reusable functions in a separate folder (usually R/), keeping the pipeline file (_targets.R) clean and organized.
- Functions defined in this file can be reused in multiple parts of the pipeline, avoiding code duplication and making it easier to maintain.

Why it's important:

- **Modularity:** Storing functions separately keeps _targets.R clean, and makes the code more reusable and maintainable.
- **Scalability:** As the project grows, having well-organized functions will make it easier to expand and update the analysis.

Project Structure for targets

main_file.R

_targets.R

functions.R

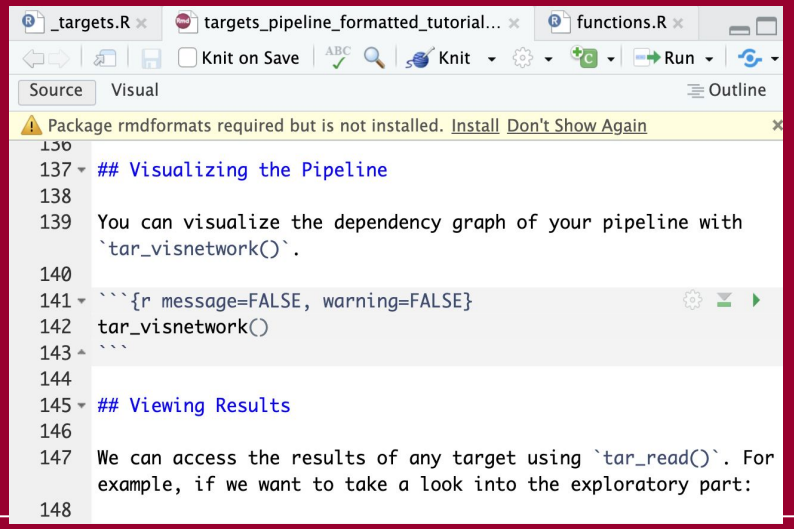
Targets and functions must be in the same name and folder structure mentioned previously.

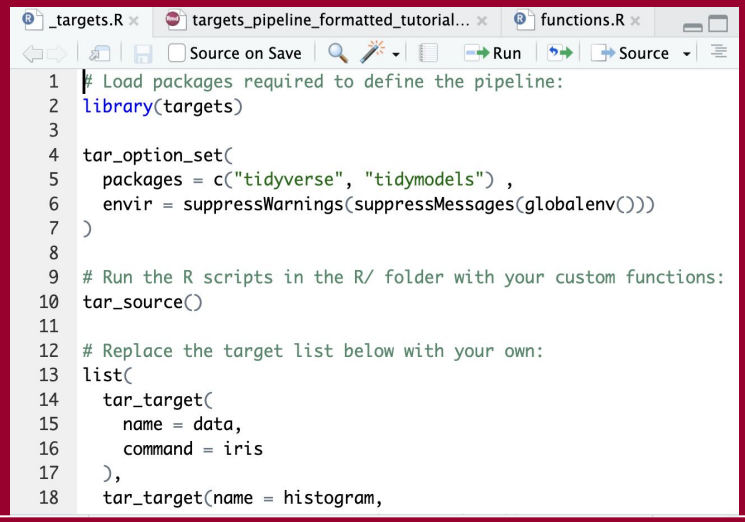
Project Structure for targets

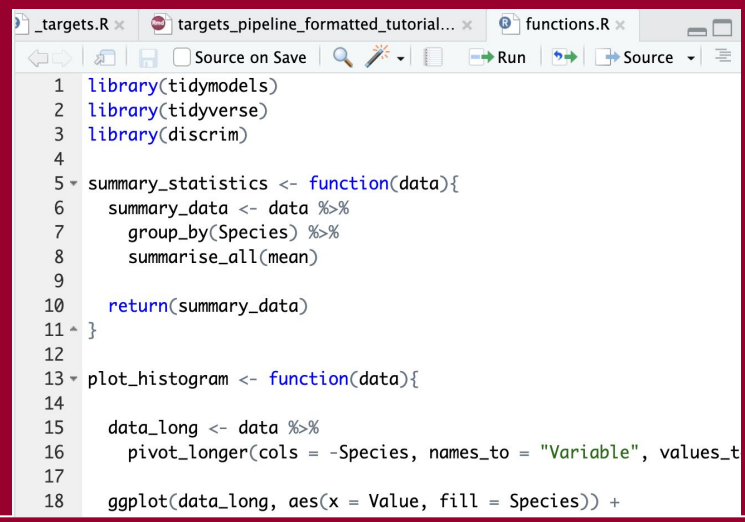
main_file.R

_targets.R

functions.R







Q&A