Data Science Worklows in R

An introduction to deploying production quality R code

Dean Marchiori

2024-01-01

Table of contents

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

```
1 + 1
```

```
[1] 2
```

1 Setup

This is a book created from markdown and executable code.

1.1 Install R

Wickham et al. (2019)

1.1.1 Install R

1.1.2 Install RStudio

1.1.3 Posit Cloud

1.2 Course Code

1.2.1 Download from Github

Course materials can be downloaded by using the following command from the `usethis` package:

```r
usethis::use_course("tbc/tbc")
```

2 Workflows

There are many approaches to tackling a data science project. Following a framework for data science workflows is important to ensuring successful delivery. The key benefits of using a data science workflow is not only ensuring the work gets done right, but it can provide a useful means for engaging stakeholders and supporters, providing project management updates and ensuring everyone is focussed on the right areas.

2.1 Data Analysis Workflows

2.1.1 CRISP-DM

CRISP-DM is a traditional framework for approaching data mining/data science/analytics projects. Developed in the 1990's it has stood the test of time and remains to popular choice today for several reasons.

Firstly, it has a strong focus on understanding the business problem and guiding the exploration of the data with subject matter experts.

Secondly, the framework provides for strict evaluation of solutions with business experts before deployment effort.

Finally, the ethos of iterative, continual improvement is built it, which aligns well with modern agile philiophies.

The key components of CRISP-DM are:

- Business Understanding
- Data Understanding
- Data Preparation
- Modelling
- Evaluation
- Deployment

A drawback of this framework is the way in which deployment is handled in modern use-cases. The need to manage the provide, hosting and monitoring of cloud or server resources has resulted in extensions to CRISP-DM.

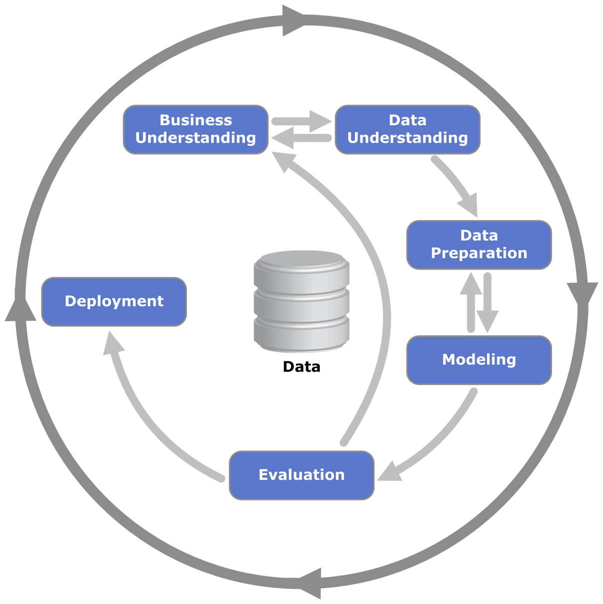


Figure 2.1: Kenneth Jensen, CC BY-SA 3.0, via Wikimedia Commons

2.1.2 Inner Loop vs Outer Loop

If we ignore the Deployment step in the CRISP-DM framework we have a nice workflow for completing 'experimental' development and modelling work.

At some point the analyst will want to deploy their work. A useful abstraction that separates the analytical work and the engineering tasks associated with deployment is through the inner loop vs outer loop concept.

The inner loop is the above mentioned CRISP-DM framework right up until deployment.

The outer loop involves the provision of computing infrastructure for model inference, model registration and versioning, deployment and endpoint provisioning, and finally monitoring and evaluation of the deployed model.

While this framework is commonly applied to deploying predictice models, adaptations can be made in the case of a dashboard, web-app or dynamic report output.

- **Outer Loop**

- Infrastructure Deployment
  - **Inner Loop**
    - Business Understanding

    * Data Understanding
    * Data Preparation
    * Modelling
    * Evaluation
- Model Registration and Deployment
- Monitoring

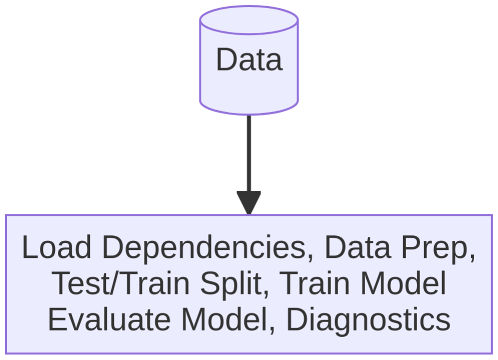
2.2 R Code Workflows

We have introduced two frameworks to think about the steps required for a successful data science project. However, these frameworks are just conceptual abstractions of course. Next we will explore commonly used R code workflows that are authored to orchestrate the end-to-end running of analysis or modelling projects - at least the inner loop components.

Once we have compared the most common approaches to translating these frameworks into code, we can evaluate how to improve strengthen our coding practices.

2.2.1 R Scripts

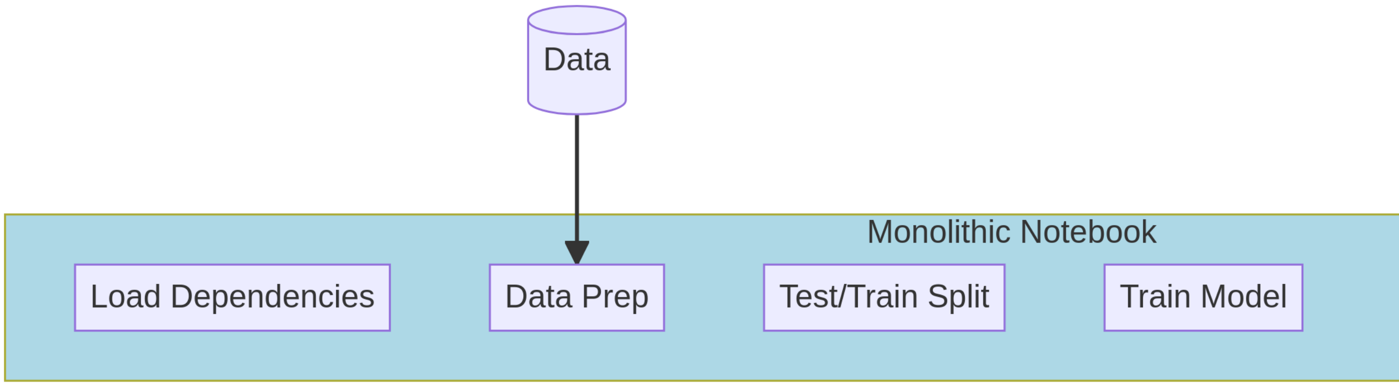
The most basic workflow is to colocate all code into a single R script. This is a common starting place for beginners or when completing small basic tasks in R. An obvious limitation is the ability to separate out logical components for testing, debugging and control flow.


Data
Load Dependencies, Data Prep, Test/Train Split, Train Model Evaluate Model, Diagnostics

2.2.2 Monolithic Markdown

A commonly adopted tool to promote more literate programming is RMarkdown or more recently quarto. These tools allow users to write plain english commentary in a markdown or visual editor and splice in ‘code-chunks’. Typically this notebook style of document is then sequentially rendered in-order and knitted into some for of output like HTML, PDF or Word etc.

This is a great way to make code more readble and self-contained while managing complexity. Obvious drawbacks exist around the execution order, control flow and caching. These can also get very long!

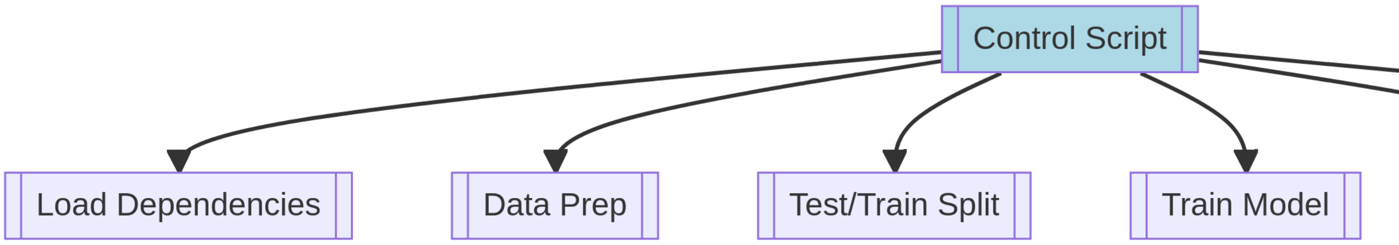


2.2.3 Control Scripts

Analysts who prefer a more scripted workflow will often attemp to break down the complexity of their project into smaller chunks, often placing parts of the analysis into their own R script.

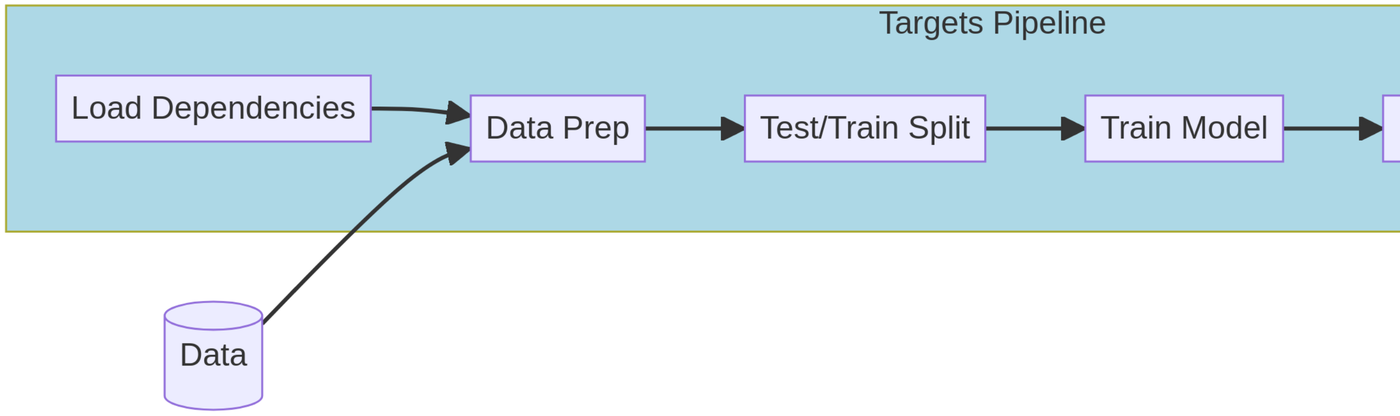
The next question is, how do we orchestrate the running of all these R scripts? This is usually solved with a ‘control’ or ‘run’ script, which `source()`’s the relevant scripts in the right order.

This is a step in the right direction, but requires lots of overhead in managing state and data flows between scripts, often by manually ‘caching’ results. The scripts are often not self-contained and this can quickly be a recipe for disaster for more complex projects.



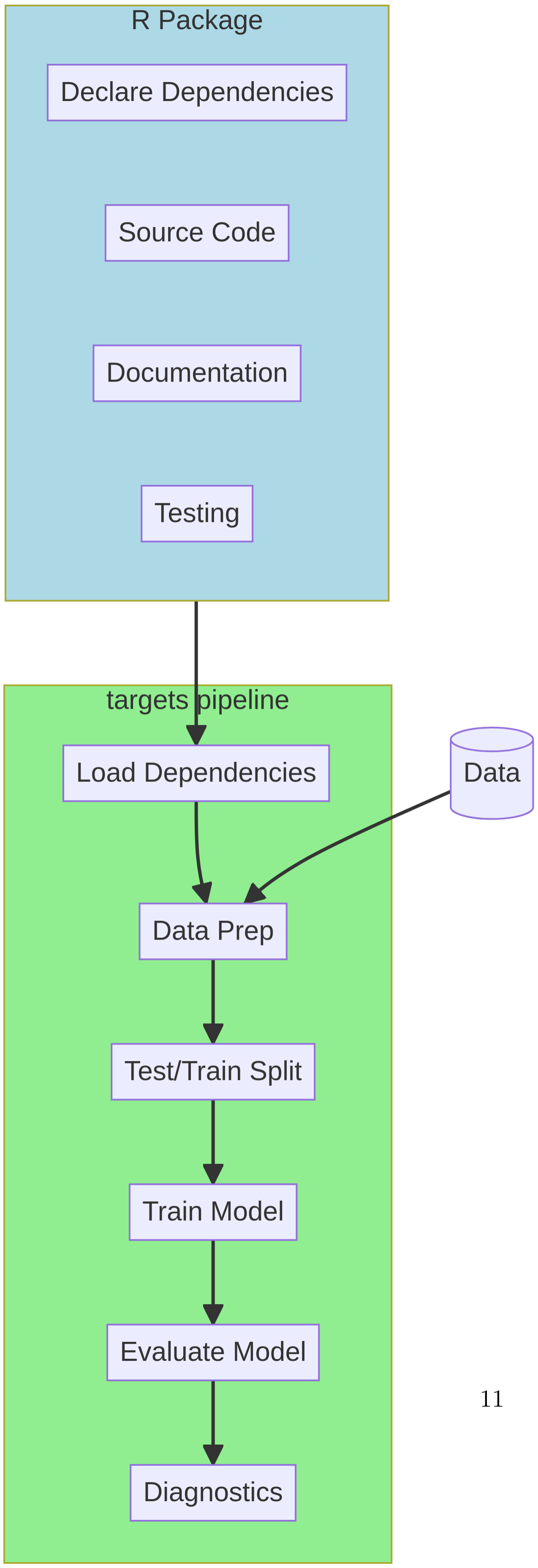
2.2.4 {targets}

{targets} is an R package that allows users to adopt a make-like pipeline philosophy for their R code. This has the advantage of more sophisticated handling of computationally-intensive workflows and provides a more opinionated structure to follow. With this are the drawbacks or forcing your collaborators to adopt the same framework and dealing with the initial learning curve.



2.2.5 R Package

An R Package is the canonical way to organise and 'package' R code for use and sharing. It provides easy means to share, install, document, test and run code.
This also follows a very opinionated structure, but unlike a third party library, it is expected knowledge for R users. This doesn't mean its easy to do! R Package's struggle to deal with the flexibility of 'doing' data analysis and are more focussed on a way to build the tools required for performing the analysis.


R Package
Declare Dependencies
Source Code
Documentation
Testing
targets pipeline
Load Dependencies
Data
Data Prep
Test/Train Split
Train Model
Evaluate Model
Diagnostics

2.3 Choosing the right workflow

So which workflow should you use?

Unfortunately this is not a straightforward decision. For quick experimental code you are unlikely to create a new R package. For a complex production deployed model, you really dont want all your code in one giant R script.

Picking the correct workflow needs to align the project goals and scope. Often this choice can evolve throughout the project.

2.3.1 An evolution

A concept or idea might be tested in a single R script, like how you would use the back of a napkin for an idea. Next you might break this down into chunks and add some prose, heading and plots so you can share and have other understand it. Next you might refactor the messy code into functions to better control the flow and the improve development practices. These functions can be documented and unit tested once you know you want to rely on them. To orchestrate the running and dependency structure to avoid re-running slow and complex code you may use the `{targets}` package. Finally to re-use, share and improve on the functions you might spin these out into their own R package!

2.3.2 Repro-retro

I talked a little about how you might want to weight and prioritise the elements of reproducibility in an rtudio::global talk in 2019. Feel free to conduct your own reproducibility-retrospective (repro-retro).

3 What is Production

3.1 Definition

- Hadley talks
- Put R in prod

3.2 Principles of Production

3.3 Patterns for R Code

3.3.1 Run Script

3.3.2 Run Shiny App

3.3.3 Run API

3.3.4 Run dashboard

4 Elements of Production Deployed R Code

- Workflow Orchestration
- CI/CD Automation
- Reproducibility
- Versioning of data, model, code
- ML metadata tracking
- Monitoring & Feedback

https://ieeexplore.ieee.org/document/10081336

Adaptation

- Orchestration
- Automation
- Reproducibility
- Version Control
- Metadata and Documentation
- Testing

4.1 Orchestration

How you structure your code

4.2 Automation

How your code is pushed/pulled

4.3 Reproducibility

4.3.1 Code dependencies

4.3.2 Packages dependencies

4.3.3 System dependencies

4.3.4 OS dependencies

4.3.5 Hardware dependencies

4.4 Version Control

Which is the current code?

4.5 Metadata and Documentation

4.5.1 Roxygen

4.5.2 R Package

4.6 Testing

4.6.1 Testthat

References

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.