Data Science Worklows in R

An introduction to deploying production quality R code

Dean Marchiori

2024-01-01

Table of contents

Preface **4**
Who is this for? . . . 4
Terminology . . . 4
Contact Me . . . 4
Contributing . . . 5
Licence . . . 5

1 Setup **6**
1.1 Software . . . 6
1.1.1 Install R . . . 6
1.1.2 Install RStudio . . . 6
1.1.3 Posit Cloud . . . 6
1.2 Code . . . 6
1.2.1 Download from Github . . . 6

2 Data Analysis Workflows **7**
2.1 Analysis Frameworks . . . 7
2.1.1 CRISP-DM . . . 7
2.1.2 Inner Loop vs Outer Loop . . . 8
2.2 R Code Workflows . . . 9
2.2.1 R Scripts . . . 9
2.2.2 Monolithic Markdown . . . 10
2.2.3 Control Scripts . . . 10
2.2.4 {targets} . . . 11
2.2.5 R Package . . . 11
2.3 Choosing the right workflow . . . 13
2.3.1 An evolution . . . 13
2.3.2 Repro-retro . . . 13

3 Development **14**
3.1 Definition . . . 14
3.2 Roles . . . 14
3.2.1 Customer . . . 15
3.2.2 Data Engineer . . . 15
3.2.3 Data Analyst . . . 15

3.2.4 Data Scientist . . . 15
3.3 Tools . . . 15

4 What is Production **17**
4.1 Definition . . . 17
4.2 Principles . . . 17
4.3 Patterns for R Code . . . 18

5 Elements of Production Deployed R Code **19**
5.1 Orchestration . . . 19
5.2 Automation . . . 20
5.3 Reproducibility . . . 20
5.3.1 Code dependencies . . . 20
5.3.2 Packages dependencies . . . 20
5.3.3 System dependencies . . . 21
5.3.4 OS dependencies . . . 21
5.3.5 Hardware dependencies . . . 21
5.4 Version Control . . . 21
5.5 Metadata and Documentation . . . 22
5.6 Testing . . . 22

6 Practical Considerations **23**
6.1 Working with IT teams . . . 23
6.2 Open Source vs Commerical Software . . . 23
6.3 Network Security . . . 23
6.4 Authentication . . . 23
6.5 Changes . . . 23

7 Case Study **24**

8 Resources **25**

9 Acknowledgements **26**

References **27**

Preface

This book is a resource for data analysts and data scientists looking to improve the way the way they write R code. While R remains a popular choice for statistical modelling and data analysis, its rapid development has enabled users to progress their work right through to being deployed into Production. However the type of work done when conducting experiments and developing models is very different to packaging up this work so it can reliably drive decisions in an organisation. This book will provide readers with an overview of contemporary frameworks for how data analysis is done in practice. It will cover how R projects are usually structures and how this can evolve based on project complexity. It will examine what is meant by experimental vs production analysis code and which principles need to be adopted. Finally it will show current tools and frameworks for taking experimental R code and strengthening it to align with best practice for reliable production grade software. Readers can step through a case study and download code to follow along.

Who is this for?

This book is intended as an introductory guide for R users who have experience writing code and fitting models, but want to improve their practices for translating these models into robust code that is reliable and used to make real-world decisions.

Terminology

Throughout this book the terms 'data science' and 'data analysis' should be considered interchangable and represent the application of advanced data analysis and statistical techniques to data to achieve an outcome. The word 'analyst' will be adopted as the primary role for someone completing these tasks.

Contact Me

If you would like to get in touch head over to deanmarchiori.com

Contributing

Contributions to this work are welcomed via Issues on the Github page.

Please note that this project uses a Contributor Code of Conduct. By contributing to this book, you agree to abide by its terms.

Licence

1 Setup

This is a book created from markdown and executable code.

1.1 Software

Wickham et al. (2019)

1.1.1 Install R

1.1.2 Install RStudio

1.1.3 Posit Cloud

1.2 Code

1.2.1 Download from Github

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

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

2 Data Analysis 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 Analysis Frameworks

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

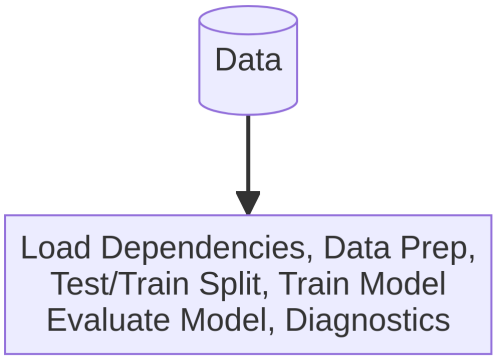
2.2 R Code Workflows

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

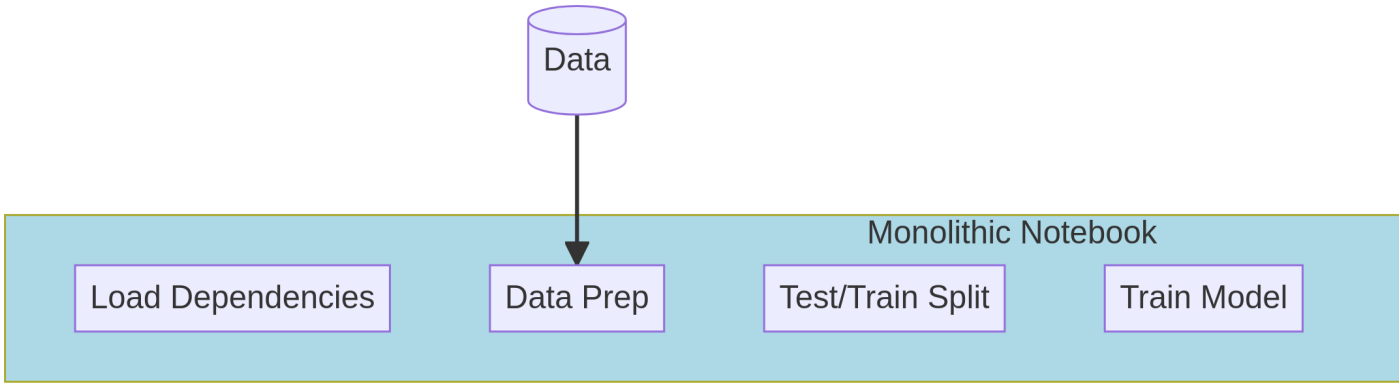
The most basic workflow is to colocate all code into a single R script. This is a common starting place for beginners or when completing small basic tasks in R. An obvious limitation is the ability to separate out logical components for testing, debugging and control flow.



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!

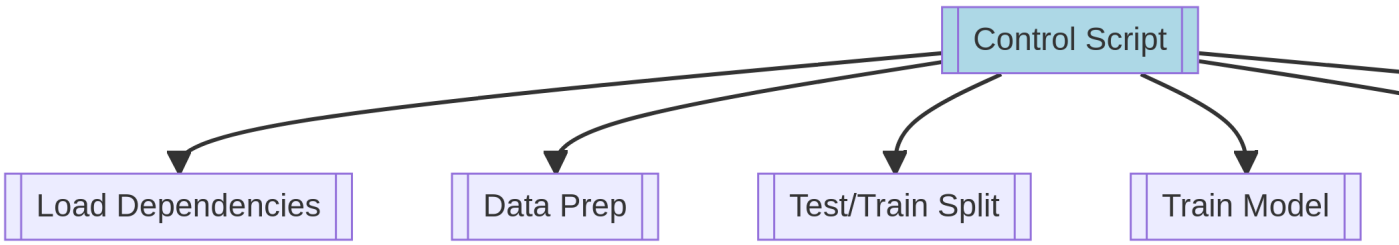


2.2.3 Control Scripts

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.

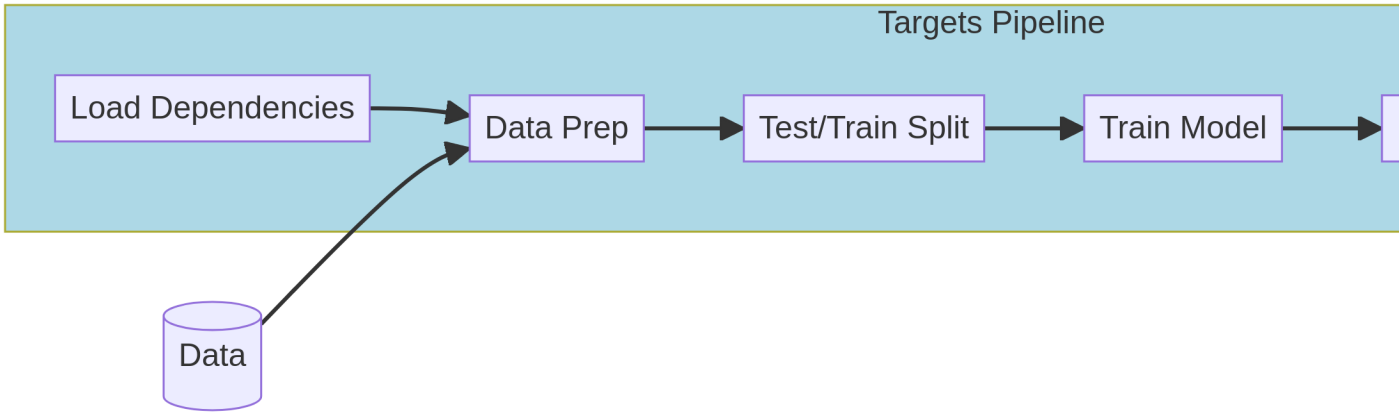
The next question is, how do we orchestrate the running of all these R scripts? This is usually solved with a 'control' or 'run' script, which `source()`'s the relevant scripts in the right order.

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.



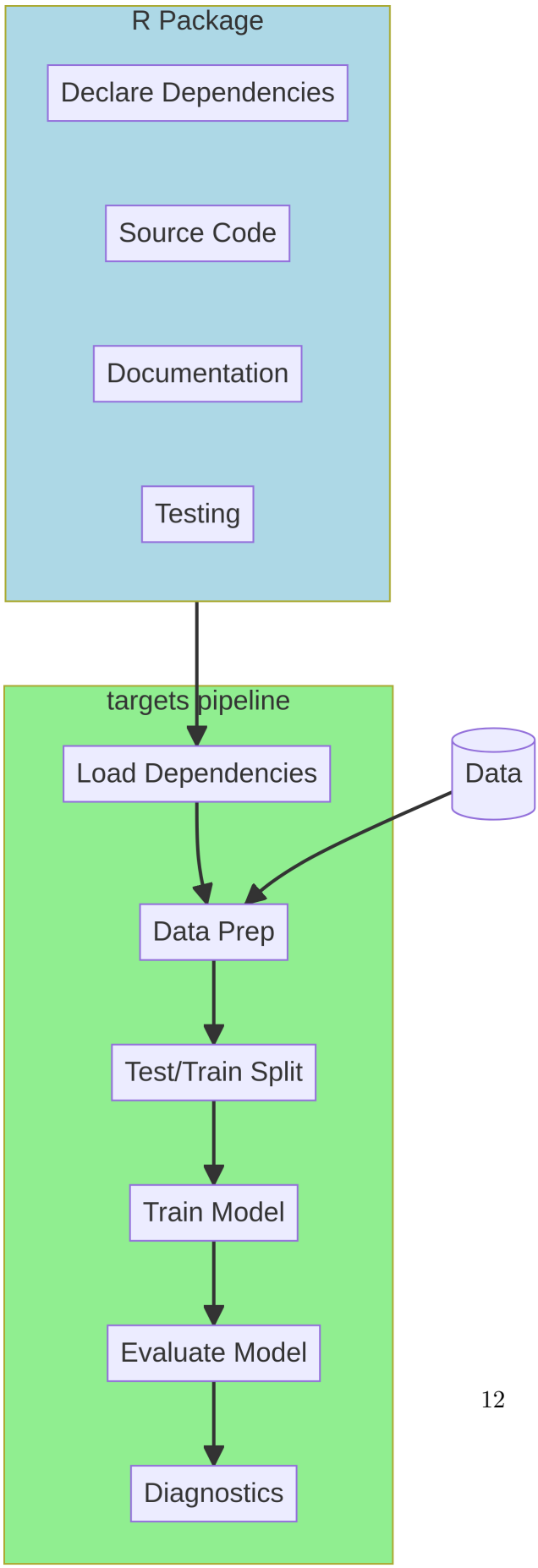
2.2.4 {targets}

{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 Development

When data science projects start, often the outcome is unknown. The work is inherently exploratory and experimental. In many cases. This results in a more, informal way of working. And, Frameworks and workflow choices to support this work. Also need to be designed differently. Definition.

3.1 Definition

A development. Is a way of working, but it is also a technical term for A hardware and software environment in many Enterprise. Data science. Organisations. Development is usually. Logical or physical separation of tools, hardware and software. That allow analysts to perform work, that is not to be relied upon.

In live decision, making Or critical infrastructure. This type of environment is prone to running. Experimental workloads tests. And day-to-day development of projects and ideas by a data science team.

A key defining feature of development code is the colocation of 'what' the code does and 'how' the code is run. That is, the functional elements of the code and the orchestration of those elements are not separated.

3.2 Roles

It can assess the various roles that take place in the development workflow. Cross-Sectionly. But looking at the full life cycle, From raw data through to a production deployed system.

In many organisations. Not all roles will be present. In some organisations, there may even be more specialised roles. For example. And ml Ops engineer May handle the deployment of machine learning workloads. While a data engineer May focus entirely on data operations, In addition a data scientist may or may not be present.

Analytical work may be done. But more generally, skilled analysts. Or even. Certain types of end users. Who were very close to the Business applications. And also have skills in data analysis.

3.2.1 Customer

Initially. A customer is a key role. Customer should be involved throughout the entire process. But in particular is active in projects at the raw data and production stage.

To contextualize and understand. And provide understanding of the raw data. And as the end user of production systems,

3.2.2 Data Engineer

In many organisations in enrichment process from raw data into some kind of data warehouse, environment is conducted by data Engineers. A data engineer may also be responsible. For provisioning infrastructure and assisting with production deployments.

3.2.3 Data Analyst

Once data is landed in a place where It is outside of front-end systems analysts can typically typically use this data to perform analysis. Business intelligence report generation. And dashboard building.

3.2.4 Data Scientist

The role of the data scientist can take many shapes. Typically it is viewed as being of most value in the stage after basic data analysis.

When more advanced modelling statistical analysis, And AI ml applications are required. In reality, it is advisable for data scientists to be involved in the raw data stage with the customer. And this aligns with, Earlier data analysis Frameworks mentioned such as crisp, DM. Where an iterative and collaborative approach is required from across all stages of the process.

[image]

3.3 Tools

The tools used in a development context. Often a subset of tools used in a wider production context. These include. Commonly thought of items such as And interactive development environment for data science such as rstudio or vs code.

Programming languages such as R or python. Version Control software. Such as git.

However, however, there are other tools. Or methodologies that can be used outside of the data, domain that are useful at this stage. These tools, facilitate the development of ideas

and use cases from stakeholders. And are often of great benefit to more technically minded analysts. To ensure that the right problem is being solved.

One example of this is the Double Diamond approach.

The Double Diamond approach is a design thinking methodology. That focuses on two key aspects first. Doing the right things. And secondly, doing the things, right? Both phases have. Two stages, the Divergence and convergence stages. In the first phase. The objective is to correctly, identify the right problem to be solved.

The Divergent stage. Can use a number of tools and facilitation techniques.

Elicit all critical business problems. And understand. Key pain points. From end users. The convergence phase is a prioritisation of these problems. To identify. The most ideal problem to be solved. This can sometimes be filtered through the lens of tools, such as desirability feasibility and viability. Once an appropriate problem has been identified to be solved similar.

Methodology can be used. To ideate Solutions. Again, it starts with a Divergence phase. Where research is conducted on available tools. And in parallel, Exploratory, data analysis is conducted.

Next, a convergence of those ideas. Is done to refine the solution. And design. The implementation. From a technical perspective.

Project on a page. Another useful technique. For engaging. With data analysis projects. Early in the life cycle, with business stakeholders is to use Frameworks such as a project on a page. This is a brief summarisation. Of the problem being solved. Some key Milestones for development. And identification of risks dependencies and other notes.

An identification for who is leading and sponsoring the project. As well as some financial analysis. If the project requires financial return on investment, As it does in many organisations. These tools while not data science Frameworks are important considerations. And are only lightly explored in this book. These will form a strong basis.

For further development, work in the right areas. Which will hopefully lead to strong business acceptance. And eventual production deployment.

4 What is Production

4.1 Definition

Production is a conceptual place where your work is being used by the intended users to make real-world decisions.

- Predictive Model integrating with front line business system via API
- Dashboard available to support user's making decisions
- A monthly report that informs management meetings
- An insight from a statistical model that has changed policy

Note how not all of the above involve a technology implementation like an API.

> Example
>
> Jenny trained a statistical model to help predict safety related incidents at worksites for her employer. One of the key factors that influenced and increase in safety incidents was significant rainfall in the previous 24 hours. The safety team has now adjusted their Safe Work checklists to include a check item for the amount of recent rainfall at that site. If its >50mm then a mandatory inspection is performed.
> This change is hardcoded in a paper form, but is still a data-science driven model in production.

4.2 Principles

For a data science project to be successully relied upon in production it should follow some key characteristics.

- Available to end users directly

- Running on stable infrastructure
- Used to make real-world decisions
- Can be replicated
- Can be reproduced safely
- Is documented sufficiently
- Can be maintained by others

In the NYR10 Conference Hadley Wickham summarises his views as:

- Not just once
- Not just my computer
- Not just me

4.3 Patterns for R Code

In terms of R specific outputs, what does a data science output look like from an R perspective.

- An R script that is run on a server on a schedule
- A `{shiny}` app hosted using on a server
- A `{plumber}` API service serving model predictions
- A hosted `{quarto}` dashboard
- An `{rmarkdown}` report
- An R package on CRAN or Github

Regardless of the solution, the fundamental concept is to get it off the developers laptop and have it be reliably available to an end user.

We will build this image up further in the next chapter when we examine more closely the elements of production deployed R code.

5 Elements of Production Deployed R Code

R has a tricky historical reputation as a programming language. With its origins in academia and commonly used in research or analysis settings it was often argued that it was not fit for production deployment. The recent development of tools to support the 'non-experimental' aspects of R coding have changed this. Below we will explore some key technical elements that should be included into any stable deployment of a data science worklow adapted from Kreuzberger, Kühl, and Hirschl (2023).

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

5.1 Orchestration

Orchestration, refers to how your code is structured and run. We explored various ways of structuring our code for data science. Workflows. In chapter two, There are two key facets to code orchestration. The first is separating the functional components. Of your project. And the orchestration of those functional components. As R is a functional programming language.

The canonical way to structure, our code is to use well-defined functions. These functions are best orchestrated as an R package. In our package. The standardised way of organising collections of R functions. As it provides a convenient means for building testing and documenting code and sharing with others. Once the functional elements of your project, Uh, stored and documented and tested appropriately.

They need to be run in the correct order. This can be achieved using a number of Frameworks in this guide. We'll be exploring the targets framework. This provides a make-like pipeline for

running R code. And automatically handles aspects such as identifying dependencies in our code. An intermediate casing of results.

5.2 Automation

Automation. Automation refers to how your project is pushed and pulled from your local development environment. Into a remote environment where users can access the results. Again, many options exist. Automation and there are varying levels of automation. Manual approaches such as click button deployment. From your IDE. Or manually copying files across to a remote server or one option.

More contemporary approaches. Involve. Continuous integration and continuous deployment practises. This is a devops style workflow. That will automatically build test. And deploy code. That has been pushed to a remote version control repository. A thorough exploration of CI, CD Solutions is beyond the scope of this book. And is now a clearly defined subspecialty known as ml Ops.

5.3 Reproducibility

5.3.1 Code dependencies

We talk about reproducibility, there are many elements of a data science workflow that need to be considered Code dependencies. The first step to a reproducible pipeline is ensuring that all users have. The same code that is being used to run the project. The recommended approach here is to ensure that all code is checked in to a remote version control repository, using a version control system such as git.

Is that why other collaborators can clone the code base from the remote git repository? And, Branch. Or Fork from the code repository in order to make their own changes, it can then be integrated back. Using proper Version Control principles. Two users running the same code may not have the same R packages installed on this system.

5.3.2 Packages dependencies

A key element of reproducibility is ensuring that all users. Can resolve. Our package dependencies this includes having the correct packages. Installing those packages from the correct locations. And ensuring the version of those packages is equivalent. A convenient solution for these problems is the RN package. The RN package does this?

5.3.3 System dependencies

System dependencies. System dependencies describe software, that is installed on. The computational environment that the project is being run on. These may include. External libraries. Such as Example. Tools like deposit, public package manager can provide some analysis of the system dependencies. That are required to support our packages on various operating systems.

Another solution explored in the next section. Is using Technologies, such as Docker. Operating system dependencies. Users, even when running the same code with the same, Our packages may find differences in how The code performs based on the operating system they're using, for example, Windows versus Linux versus Mac OS.

5.3.4 OS dependencies

It is common in a production setting to deploy code. To an external server using. A Linux operating system. A way to control operating system and system dependencies. Is. Use Technologies such as Docker, Docker does this. A basic Docker file is shown below. Finally. The dependencies.

5.3.5 Hardware dependencies

Hardware dependencies are a little trickier. These are physical Hardware infrastructure constraints on how a project is run. Regardless of the operating system and software that is running on it. For example, this is commonly seen with the use of CPU versus GPU Technologies and different types of processor chips. A thorough exploration of Hardware dependencies is outside. The scope of this guide.

5.4 Version Control

Version Control. Version Control is a critical aspect. To ensure. Reproducibility instability in production deployed data, science Solutions. Version control includes. Not only Version Control for code, but also Version Control for data and models. Version control for code is commonly achieved using the get Tool.

Git is a version control system that allows users to Do this. If the data science project results in a Statistical and machine learning model being fit. It is the model itself that will be the deployed artefact in order for inference to be performed. Therefore it is critical that this model B also versioned. Again, there are many solutions for this. However, a recent development in, the r ecosystem is the vetivert package. A bit of a package, does this? The exploration of Version Control with data is beyond the scope of this book. However, It is recommended.

That data is also versioned and stored appropriately in a secured data store, such as a data Lake or data warehouse.

5.5 Metadata and Documentation

Metadata and documentation. It is important. When deploying artefacts into a production setting, that there is appropriate metadata. Metadata refers to. Tags versions dates and other relevant information to describe what work is being deployed. We'll explore this further. In a modelling context, using the bit of a package. In terms of metadata and documentation for the functional aspects of the code, We can rely on the Internal documentation used in our packages.

Based on the oxygen package. This provides a useful template for documenting, our functions using oxygen tags. That are automatically generated into help documents. Another form of documentation in an R package is a readme. The package readme is an important artefact to tell users, what? Software does how it is to be run in any other important information such as the licence or prerequisites or dependencies.

Also how to contribute and get help? Low form documentation.

5.6 Testing

Testing. Finally. Test unit testing is important. Element of software engineering. In terms of our code. Testing, can be performed using the test that package. The test that package does this. And here is an example of it. Conveniently the test. That package is integrated into the build process for our packages.

Using the L Studio IDE. Testing can also be performed on a directory. Using this function.

6 Practical Considerations

6.1 Working with IT teams

- Install
- Admin
- Maintainence
- Debugging

6.2 Open Source vs Commerical Software

6.3 Network Security

6.4 Authentication

6.5 Changes

- Model drift
- New features

7 Case Study

8 Resources

Link to github

9 Acknowledgements

References

Kreuzberger, Dominik, Niklas Kühl, and Sebastian Hirschl. 2023. "Machine Learning Operations (MLOps): Overview, Definition, and Architecture." *IEEE Access* 11: 31866–79. https://doi.org/10.1109/ACCESS.2023.3262138.

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.