Simple process for sharing R code via Docker

Ronald (Ryy) Glenn Thomas

2025-03-10

This white paper presents a comprehensive approach to achieving reproducibility in R workflows by combining two powerful tools: renv for R package management and Docker for containerizing the computing environment. Together, these tools ensure that an R workflow runs identically across different systems with the same packages, R version, and system libraries as the original setup. The paper includes a practical case study demonstrating collaborative development using this approach.

Table of contents

1 Motivation

We have code we want to send to a colleague to run and/or edit. This can be a more difficult task than we might think at first. In order for our colleague to have a straightforward set of steps to take in order for the code to run in a error-free and reporducible way we need to avoid several potential problems; the problem

of potentially different `R` and `R`-package versions. We also need to avoid needing to install support files like `pandoc` and `latex` as well as supplemental files called by the program, e.g. `bibtex` reference files, `latex` preamble files, or data sets in csv or other formats, graphic images, etc.

To insure that will be the case we will follow a procedure that will assure our process is glitch-free and reproducility goals are met. Although more involved than simply relaying the R file via e.g. email, the approach we'll describe here is, in our opinion, well worth the effort. effective.

When you finish this post you or your colleague will be able to simply pull a custom built image from `dockerhub`, say rgt47/penguins1, and run the command:

```
docker run  -v "$(pwd):/home/joe" -v "$(pwd)/output:/home/joe/output" rgt47/penguins1
```

to have a full R environment installed on your desktop identical to the developers for your codebase. Then navigating to the R directory in the repo run:

```
git branch code_edits
git checkout code_edits
```

```
vim peng1.Rmd
```

To open the R code for editing

and

```
R -e "library(rmarkdown); render('peng1.Rmd')"
```

to process code through `R`, `pandoc`, and `latex`.

When edits are complete, first run any available tests and add additional tests for the additions made to the code:

```
> devtools::test()
> devtools::testthat()
```

2 Introduction

Its often the case that two or more research collaborators want to work on the `R` codebase for the same project. To accomplish this, unfortunately, its often not as simple as one researcher sending a set of text files containing the code to the other colleagues.

For example, a number of elements of the computing environment can differ between collaborators' computers, for example, the version of `R`, the operating system (macos vs. windows vs linux), the `R` packages installed, the versions of these `R` packages, `R` startup files (e.g. `.Rprofile`, and `.Renviron`). Any of these components of a researchers computing environment can cause code working for one reseracher to break when run by a second.

3 Methods

Lets assume you have some `R` program code in a file, say `peng.R`, that you're written to analyze some "Palmer Penguin" data. You want to share the code with a colleague, we'll call him Joe. How to proceed?

The simplest option is simply to send Joe the `R` file containing the code via the most convenient method (e.g. email)

The next step is for for Joe to (attempt to) run the code with the idea of expanding the analysis. Typically he would do this either using an IDE, such as `Rstudio`, or run the code from the command line in the terminal with the command:

```
> R -e "source('peng1.Rmd')"
```

Sometimes this approach works. When it does, Joe can add comments or expand the code and relay it back to you, "rince and repeat" and all is well. Frequently, however, this simple approach will fail for any of several reasons. Even when it runs, unless care is taken, its not guaranteed that Joe will get the same results you do.

Ideally to facilitate reproducibility your colleague Joe will have a computing environment as similar to yours as possible. This can be difficult to achieve, especially given the dynamic nature of open source software. For example, Joe may have an older version of `R` installed on his workstation, or his `R` environment may be missing a necessary package to run the code or it may be the wrong version. Additiional potential problems include: the program may need to source an additional file or load data that's missing.

All of these problems go away if instead of sending the program as a standalone text file you send it as a docker image. In this post we'll walk through the process of dockerizing the R code.

4 Methods

Assume we have a simple `R` file that we want to share with Joe such as the following:

```
---
title: "\\includegraphics[width=2cm, right]{sudoku_apple.pdf}\\newline
  New Penguin plot"
author: "R.G. Thomas"
date: "`r format(Sys.time(), '%B %d, %Y')`"
fontsize: 11pt
geometry: "left=3cm,right=5cm,top=2cm,bottom=2cm"
output:
  pdf_document:
    keep_tex: true
    includes:
header-includes:
   - \usepackage{lipsum}
   - \usepackage[export]{adjustbox}
   - \usepackage{currfile}
   - \usepackage{fancyhdr}
   - \pagestyle{fancy}
   - \setlength{\headheight}{14pt}
   - \fancyfoot[L]{\currfilename} %put date in header
   - \fancyfoot[R]{\includegraphics[width=.8cm]{sudoku_apple.pdf}}
   - \fancyhead[L]{\today} %put current file in footer
```

````
   - \fancyhead[R]{Penguins RGT Lab report}
latex_engine: xelatex
---

```{r  echo=F}
# clear env: objects and packages
library(pacman)
p_load(palmerpenguins, rmarkdown, knitr)

opts_chunk$set(
  warning = FALSE, message = FALSE, echo = FALSE, fig.width = 5.2,
  fig.height = 3, results = "asis", dev = "pdf"
)
```

# Introduction

\lipsum[1-5]

Draft report examining Penguin characteristics. Data from Alison Horst article
in the `R journal`
@m.horstPalmerArchipelagoPenguins2022

```{r }
attach(penguins)
plot(flipper_length_mm, bill_length_mm, col = sex)
legend("topleft", legend = levels(sex), pch = 16, col = unique(sex))
```

\lipsum[1-5]
\bibliography{penguins.bib}
\bibliographystyle{plain}
\nocite{*}
````

5 Share program code with Joe.

Joe downloads the attachment. Opens a working directory and attempts to run the Rmd file

with the command

```
> R -e "source('peng1.Rmd')"
```

Joe has a linux mint desktop

```
> mkdir peng1
> cd peng1
> R -e "render('peng1.Rmd')"
```

Linux can't find R

Joe can fix this by installing `R`

```
> sudo apt install r-base-core
```

So install R…. to fix

> sudo apt install r-base-core Try again…
>
> R -e "render('peng1.Rmd')" Result: R loads but gives error: could not find function "render"

Search google "R render"

Result: https://pkgs.rstudio.com/rmarkdown/reference/render.html

Looks like render is a function inside rmarkdown package. so install package

```
R> install.packages("rmarkdown")
```

Try again.

```
> R -e "library(rmarkdown); render('peng1.Rmd')"
```

Result: error. pandoc version 1.12.3 or higher required.

Now try to install pandoc Try again. > R -e "library(rmarkdown); render('peng1.Rmd')" Result: error. there is no called pacman

so install pacman R> install.packages("pacman") Try again…

R -e “library(rmarkdown); render(‘peng1.Rmd’)”
result: failed pandoc could not find /Users/zenn/shr/preamble.tex

This makes sense we forgot to send the preamble.tex file to researcher 2. Lets send it now. Also we need to adjust its location from a macos style file name to a linux based one.

To edit peng1.Rmd we need vim > sudo apt install vim

Try again Result: pandoc error “pdflatex not found” Lets install tinytex. First in R and then in mint R> install.packages(“tinytex”) R> tinytex::install_tinytex() # to uninstall TinyTeX, run # tinytex::uninstall_tinytex()

Try again:

R -e “library(rmarkdown); render(‘peng1.Rmd’)”
result: pandoc error: file sudoku_apple.pdf not found.

This makes sense we forgot to send the logo file. Lets send the file and try again.

R -e “library(rmarkdown); render(‘peng1.Rmd’)”
Result: error no bibliography file found.

This makes sense we forgot to send the bib file. Lets send the file and try again. Also need to edit the location of bib file.

Try again:

R -e “library(rmarkdown); render(‘peng1.Rmd’)”

results: minor error. Some packages weren’t loading via pacman.

Try removing janitor, kableExtra, tidyverse, readxl and add ggplot2.

Try again:

R -e “library(rmarkdown); render(‘peng1.Rmd’)”

Success!

Lesson learned: The program peng1.Rmd needs access to three external files: 1. preamble.tex 2. penguins.bib 3. sudoku_apple.pdf

6 Docker approach

Alternatively, consider the "Docker" approach.

Before sending peng.Rmd to Joe we'll dockerize it.

- Prepare a work directory: peng1. We want to prepare a repo and a docker image that has a standardized computing environment set up including R, the R packages and all the preliminaries taken care of, so that all Joe has to do is clone the repo and run the image from dockerhub.

Steps for Joe:

- install docker on his machine.
- download docker image from dockerhub
- run image

Here is the docker file

```
FROM rocker/r-devel
RUN apt-get update -qq && apt-get -y --no-install-recommends install \
    libxml2 \
    libxml2-dev \
    libcairo2-dev \
    libsqlite3-dev \
    libpq-dev \
    libssh2-1-dev \
    unixodbc-dev \
    r-cran-v8 \
    libv8-dev \
    net-tools \
    libprotobuf-dev \
    protobuf-compiler \
    libjq-dev \
```

```
    libudunits2-0 \
    libudunits2-dev \
    libgdal-dev \
    libssl-dev
## update system libraries
RUN apt-get update && \
    apt-get upgrade -y && \
    apt-get clean
RUN apt-get install pandoc -y
RUN groupadd --system joe
RUN useradd --system --gid joe -m joe
RUN chown joe:joe -R /home/joe
USER joe
WORKDIR /home/joe
COPY renv.lock renv.lock
RUN R -e "install.packages('renv', repos = c(CRAN = 'https://cloud.r-project.org'))"
COPY renv/settings.json renv/settings.json
RUN R -e "renv::restore()"
CMD ["/bin/bash"]
```

run docker

```
docker build -t rgt47/penguins1 --platform=linux/amd64 .
```

relay image to Joe via dockerhub.

```
docker push rgt47/penguins1
```

or

```
docker save rgt47/penguins1 | gzip > penguins1_transfer.tgz
docker load -i penguins1_transfer.tgz
```

```
> docker pull rgt47/penguins1

docker run -it --rm \
-v "$(pwd):/home/joe" -v "$(pwd)/output:/home/joe/output" rgt47/penguins1
```

```
> cd output
> library(rmarkdown); render('peng1.Rmd')
```

Important to include the association between the /home/joe/output directory in the container with the output directory on the local workstation. Thats where the results of the analysis will be saved.

```
> R -e "library(rmarkdown); render('peng1.Rmd')"
```

and if he wants to edit peng.Rmd

```
> vim peng1.Rmd
```

7 REFERENCES

Running your R script in Docker

```
docker run --rm -p 8787:8787 -e PASSWORD=z rocker/rstudio

j share
cd peng1

docker build -t rgt47/penguin1 .
docker run --rm -it rgt47/penguin1

droot="${PWD}"

docker run -it --rm -v $droot:/home/joe/output rgt47/penguin1

Problem
  docker run -it --rm -v $droot:/home/joe rgt47/penguins1

Unable to find image 'rgt47/penguins1:latest' locally
docker: Error response from daemon: manifest for rgt47/penguins1:latest not found: manifest unk
See 'docker run --help'.
share_R_code_via_docker/peng1   echo $droot
/Users/zenn/prj/qblog/posts/share_R_code_via_docker/peng1
```

```
share_R_code_via_docker/peng1   mkdir output
share_R_code_via_docker/peng1   f

Need a base R that includes dev tools

Try rocker/r-devel

docker build -t rgt47/penguin2 . --platform linux/amd64


docker run --rm \
  -e DISABLE_AUTH=true -e ROOT=true \
  -v "$(pwd):/home/rstudio/project:rw" \
  -p 8787:8787 \
  --name rstudio rocker/verse:3.6.3

docker run -it --rm -v "$(pwd):/home/joe" rgt47/penguins1
```

Dockerfile with R code

```
FROM rocker/r-devel
RUN apt-get update && \
    apt-get upgrade -y && \
    apt-get clean
RUN apt-get install terminator tree eza ssh zsh git fzf ripgrep vim curl \
autojump  zsh-autosuggestions sudo pandoc -y
RUN Rscript -e 'install.packages("renv")'
COPY renv.lock renv.lock
RUN  Rscript -e 'renv::restore()'
RUN groupadd --system joe
RUN useradd --system --gid joe -m joe
RUN usermod -aG sudo joe
RUN echo joe:z | chpasswd
RUN chown joe:joe -R /home/joe
RUN chown joe:joe -R /usr/local/lib/R/site-library
RUN chsh -s $(which zsh) joe
WORKDIR /home/joe/
COPY .vimrc .vimrc
COPY .zshrc .zshrc
```

```
RUN chown joe:joe -R /home/joe
RUN mkdir -p  /home/joe/output
USER joe
RUN curl -fLo ~/.vim/autoload/plug.vim --create-dirs \
    https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim
CMD ["/bin/zsh"]
```

```
docker build -t rgt47/penguins1 . --platform linux/amd64
docker pull rgt47/penguins
docker run -it --rm -v "$(pwd):/home/joe/peng2" -v "$(pwd)/out:/home/joe/output" rgt47/penguins
```

7.1 Executive Summary

Reproducibility stands as a cornerstone of professional data analysis, yet in practice, achieving it consistently with R workflows remains challenging. R projects frequently break when transferred between computers due to mismatched R versions or package dependencies, leaving developers in what is colloquially known as "dependency hell." This white paper presents a comprehensive approach to solving this problem by combining two powerful tools: **renv** for R package management and **Docker** for containerizing the computing environment. Together, these tools ensure that an R workflow runs identically across different systems with the same packages, R version, and system libraries as the original setup.

7.2 Introduction

7.2.1 The Challenge of Reproducibility in R

R has become a standard tool for data science and statistical analysis across numerous disciplines. However, as R projects grow in complexity, they often develop intricate webs of dependencies that can make sharing and reproducing analyses difficult. Some common challenges include:

- Different R versions across machines
- Incompatible package versions
- Missing system-level dependencies

- Operating system differences
- Conflicts with other installed packages

These challenges often manifest as the frustrating "it works on my machine" problem, where analysis code runs perfectly for the original author but fails when others attempt to use it. This undermines the scientific and collaborative potential of R-based analyses.

7.2.2 A Two-Level Solution

To address these challenges comprehensively, we need to tackle reproducibility at two distinct levels:

1. **Package-level reproducibility**: Ensuring exact package versions and dependencies are maintained
2. **System-level reproducibility**: Guaranteeing consistent R versions, operating system, and system libraries

The strategy presented in this white paper leverages **renv** for package-level consistency and **Docker** for system-level consistency. When combined, they provide a robust framework for end-to-end reproducible R workflows.

7.3 renv: Package-Level Reproducibility

7.3.1 What is renv?

renv (Reproducible Environment) is an R package designed to create isolated, project-specific library environments. Instead of relying on a shared system-wide R library that might change over time, renv gives each project its own separate collection of packages with specific versions.

7.3.2 Key Features of renv

- **Isolated project library**: renv creates a project-specific library (typically in `renv/library`) containing only the packages used by that project. This isolation ensures that

updates or changes to packages in one project won't affect others.

- **Lockfile for dependencies**: When you finish installing or updating packages, `renv::snapshot()` produces a `renv.lock` file - a JSON document listing each package and its exact version and source. This lockfile is designed to be committed to version control and shared with collaborators.
- **Environment restoration**: On a new machine (or when reproducing past results), `renv::restore()` installs the exact versions of packages specified in the lockfile. This creates an R package environment identical to the one that created the lockfile, provided the same R version is available.

7.3.3 Basic renv Workflow

The typical workflow with renv involves:

```
# One-time installation of renv
install.packages("renv")

# Initialize renv for the project
renv::init()  # Creates renv infrastructure

# Install project-specific packages
# ...

# Save the package state to renv.lock
renv::snapshot()

# Later or on another system...
renv::restore()  # Restore packages from renv.lock
```

While renv effectively handles package dependencies, it does not address differences in R versions or system libraries. This limitation is where Docker becomes essential.

7.4 Docker: System-Level Reproducibility

7.4.1 What is Docker?

Docker is a platform that allows you to package software into standardized units called containers. A Docker container is like a lightweight virtual machine that includes everything needed to run an application: the code, runtime, system tools, libraries, and settings.

7.4.2 Docker's Role in Reproducibility

While renv handles R packages, Docker ensures consistency for:

- **Operating system**: The specific Linux distribution or OS version
- **R interpreter**: The exact R version
- **System libraries**: Required C/C++ libraries and other dependencies
- **Computational environment**: Memory limits, CPU configuration, etc.

By running an R Markdown project in Docker, you eliminate differences in OS or R installation as potential sources of irreproducibility. Any machine running Docker will execute the container in an identical environment.

7.4.3 Docker Components for R Workflows

For R-based projects, a typical Docker approach involves:

1. **Base image**: Starting from a pre-configured R image (e.g., from the Rocker project)
2. **Dependencies**: Adding system and R package dependencies
3. **Configuration**: Setting working directories and environment variables
4. **Content**: Adding project files
5. **Execution**: Defining how the project should run

A simple Dockerfile for an R Markdown project might look like:

```
# Use R 4.1.0 on Linux as base image
FROM rocker/r-ver:4.1.0

# Set the working directory inside the container
WORKDIR /workspace

# Install renv and restore dependencies
RUN R -e "install.packages('renv', repos='https://cloud.r-project.org')"

# Copy renv lockfile and infrastructure
COPY renv.lock renv/activate.R /workspace/

# Restore the R package environment
RUN R -e "renv::restore()"

# Default command when container runs
CMD ["/bin/bash"]
```

This Dockerfile creates a consistent environment with a specific R version and packages, regardless of the host system.

7.5 Combining renv and Docker: A Comprehensive Approach

7.5.1 Why Use Both?

Using renv or Docker alone improves reproducibility, but combining them provides the most comprehensive solution:

- **Docker** guarantees the OS and R version
- **renv** guarantees the R packages and their versions
- **Together** they achieve end-to-end reproducibility from operating system to package dependencies

This combined approach creates a fully portable analytical environment that can be shared and will produce identical results across different computers.

7.5.2 Integration Strategy

The recommended workflow integrates renv and Docker in the following manner:

1. **Develop locally with renv**: Create your R project with renv to manage package dependencies.
2. **Snapshot dependencies**: Use `renv::snapshot()` to create a lockfile.
3. **Containerize with Docker**: Create a Dockerfile that uses a specific R version and incorporates the renv lockfile.
4. **Share both**: Distribute both the code (with lockfile) and the Docker configuration.
5. **Execute consistently**: Run analyses in the Docker container for guaranteed reproducibility.

This strategy ensures that your R Markdown documents and analyses will run identically for anyone who has access to your Docker container, regardless of their local setup.

7.6 Practical Example: Collaborative R Markdown Development

The following case study demonstrates how two developers can collaborate on an R Markdown project using renv and Docker to ensure reproducibility.

7.6.1 Project Scenario

Two data scientists are collaborating on an analysis of the Palmer Penguins dataset. Developer 1 will set up the initial project structure and create a basic analysis. Developer 2 will extend the analysis with additional visualizations. They'll use GitHub for version control and DockerHub to share the containerized environment.

7.6.2 Step-by-Step Implementation

7.6.2.1 Developer 1: Project Setup and Initial Analysis

Step 1: Create and Initialize the GitHub Repository

Developer 1 creates a new GitHub repository called "penguins-analysis" and clones it locally:

```
git clone https://github.com/username/penguins-analysis.git
cd penguins-analysis
```

Step 2: Initialize renv for Dependency Management

```
install.packages("renv")  # If not already installed
renv::init()  # Initialize renv for the project
```

This creates the necessary renv infrastructure, including an initial `renv.lock` file.

Step 3: Install Required R Packages

```
install.packages("ggplot2")
install.packages("palmerpenguins")
renv::snapshot()  # Save package versions to renv.lock
```

Step 4: Create Initial R Markdown Analysis

Developer 1 creates a file named `peng1.Rmd` with the following content:

7.7 Flipper Length vs. Bill Length

```
#| label: flipper-bill-plot
ggplot(penguins, aes(x = flipper_length_mm, y = bill_length_mm)) +
  geom_point() +
  theme_minimal() +
  ggtitle("Flipper Length vs. Bill Length")
```

Listing 1 peng1.Rmd

````
---
title: "Palmer Penguins Analysis"
author: "Developer 1"
date: "2025-03-10"
output: html_document
---

```r
#| label: setup
#| include: false
library(ggplot2)
library(palmerpenguins)
````

Step 5: Create a Dockerfile

Developer 1 creates a Dockerfile that deliberately excludes the R Markdown file to ensure that

```{.dockerfile filename="Dockerfile"}
# Use R 4.1.0 as base image
FROM rocker/r-ver:4.1.0

# Set the working directory inside the container
WORKDIR /workspace

# Install renv and restore dependencies
RUN R -e "install.packages('renv', repos='https://cloud.r-project.org')"

# Copy only the renv.lock and renv infrastructure
COPY renv.lock renv/activate.R /workspace/

# Restore the R package environment
RUN R -e "renv::restore()"

CMD ["/bin/bash"]
```

Step 6: Build and Push the Docker Image
```

```
docker build -t username/penguins-analysis:v1 .
docker login
docker push username/penguins-analysis:v1
```

Step 7: Commit and Push to GitHub

Developer 1 commits the project files (excluding the R Markdown document from the Docker image):

```
git add .
git commit -m "Initial renv setup and Docker environment (without Rmd)"
git push origin main
```

Step 8: Communicate with Developer 2

Developer 1 provides these instructions to Developer 2:

1. Clone the GitHub repository
2. Pull the prebuilt Docker image from DockerHub
3. Run the container interactively, mounting the local repository
4. Extend the analysis in the peng1.Rmd file
5. Push changes back to GitHub

7.7.0.1 Developer 2: Extending the Analysis

Step 1: Clone the Repository and Pull the Docker Image

```
git clone https://github.com/username/penguins-analysis.git
cd penguins-analysis
docker pull username/penguins-analysis:v1
```

Step 2: Run Docker Interactively

Developer 2 runs the container with the local repository mounted:

```
docker run --rm -it -v "$(pwd):/workspace" -w /workspace username/penguins-analysis:v1 /bin/bas
```

This approach: - Uses the renv-restored environment from the container - Allows Developer 2 to access and modify files directly from their local machine

Step 3: Extend the Analysis

Developer 2 modifies `peng1.Rmd` to add a second plot for body mass vs. bill length:

Listing 2 `peng1.Rmd (modified)`

````
---
title: "Palmer Penguins Analysis"
author: "Developer 2"
date: "2025-03-10"
output: html_document
---

```r
#| label: setup
#| include: false
library(ggplot2)
library(palmerpenguins)
````

7.8 Flipper Length vs. Bill Length

```
#| label: flipper-bill-plot
ggplot(penguins, aes(x = flipper_length_mm, y = bill_length_mm)) +
  geom_point() +
  theme_minimal() +
  ggtitle("Flipper Length vs. Bill Length")
```

7.9 Body Mass vs. Bill Length

```
#| label: mass-bill-plot
ggplot(penguins, aes(x = body_mass_g, y = bill_length_mm)) +
  geom_point() +
```
```

```
  theme_minimal() +
  ggtitle("Body Mass vs. Bill Length")
```

```
**Step 4: Commit and Push Changes Back to GitHub**

```bash
git add peng1.Rmd
git commit -m "Added second plot: Body Mass vs. Bill Length"
git push origin main
```

7.9.1 Key Benefits Demonstrated in This Example

This collaborative workflow demonstrates several advantages of the renv + Docker approach:

1. **Dependency consistency**: Both developers work with identical R package versions thanks to renv.
2. **Environment consistency**: The Docker container ensures the same R version and system libraries.
3. **Separation of concerns**: The R Markdown document remains outside the Docker image, allowing for easier collaboration.
4. **Workflow flexibility**: Developer 2 can work in the container while editing files locally.
5. **Full reproducibility**: The entire analysis environment is captured and shareable.

7.10 Best Practices and Considerations

7.10.1 When to Use This Approach

The renv + Docker approach is particularly valuable for:

- **Long-term research projects** where reproducibility over time is crucial
- **Collaborative analyses** with multiple contributors on different systems
```

- **Production analytical pipelines** that need to run consistently
- **Academic publications** where methods must be reproducible
- **Teaching and education** to ensure consistent student experiences

7.10.2 Tips for Efficient Implementation

1. **Keep Docker images minimal**: Include only what's necessary for reproducibility.
2. **Use specific version tags**: For both R packages and Docker base images, specify exact versions.
3. **Document system requirements**: Include notes on RAM and storage requirements.
4. **Leverage bind mounts**: Mount local directories to containers for easier development.
5. **Consider computational requirements**: Particularly for resource-intensive analyses.

7.10.3 Potential Challenges

Some challenges to be aware of:

- **Docker image size**: Images with many packages can become large
- **Learning curve**: Both Docker and renv require some initial learning
- **System-specific features**: Some analyses may rely on hardware features
- **Performance considerations**: Containers may have different performance characteristics

7.11 Conclusion

Achieving full reproducibility in R requires addressing both package dependencies and system-level consistency. By combining renv for R package management and Docker for envi-

ronment containerization, data scientists and researchers can create truly portable and reproducible workflows.

This approach ensures that the common frustration of "it works on my machine" becomes a thing of the past. Instead, R Markdown projects become easy to share and fully reproducible. A collaborator or reviewer can launch the Docker container and get identical results, without worrying about package versions or system setup.

The case study presented demonstrates how two developers can effectively collaborate on an analysis while maintaining reproducibility throughout the project lifecycle. This strategy represents a best practice for long-term reproducibility in R, meeting the high standards required for professional data science and research documentation.

By adopting this two-tool approach, the R community can make significant strides toward the goal of fully reproducible research and analysis.

7.12 References

1. Thomas, R.G. "Docker and renv strategy."
2. "Palmer Penguins Analysis." Developer 2.
3. The Rocker Project. https://www.rocker-project.org/
4. renv documentation. https://rstudio.github.io/renv/