Bono Usu

A guide for good practice in R programming

Jon Calder & Lorenz Walthert

Contents

Preface

Bono usu is Latin for *good practice.* The aim of this book is to outline some good practice for R programming.

Chapter 1

Introduction

Welcome to *bono usu*, a book about good practice for R programming. This book tries to compile and further build upon resources on the topic of good practice in R programming in a somewhat broad sense. We acknowledge that there is no clear 'best practice' when it comes to many aspects of R code style, workflow etc, but there are certainly some principles which can direct one towards 'better practice'.

When collaborating with others on code projects it is hugely beneficial (if not essential) to agree on and abide by some style guidelines and workflow in order to facilitate the process of collaboration. The aim of this book is to bring together some popular opinion, convention, and/or suggested practice which has helped to inform us in facilitating our own consensus on 'good practice'.

The book covers the following topics:

- Code style guide: This chapter is about low level style use in R.
- Workflow & collaboration: In this chapter, we introduce some commonly used workflow tools, and explore how one can make the most of them, while taking a broader perspective on high-level practices in the organisation of projects.
- Package development: Finally, we address the topic of developing extensions for R.

Chapter 2

Code style guide

This chapter deals mostly with low-level coding style. Every section starts with a code chunk to quickly highlight the principle of that section. Depending on how strongly we agree/disagree with a certain practice, we use `# good / bad` or `# advised / discouraged` to indicate the importance of adhering to a certain style.

After the summary, the pros and cons of certain approaches are discussed more in depth.

2.1 Assignment

```
# good
a <- 2
# bad
a = 2
```

Although programmers coming from other languages may not immediately see the benefit of using the so called *assignment operator* instead of the equals sign, the main advantage is that that an *asignment context* can be clearly distinguished from other contexts, for example from a function call and from a comparison of two objects:

```
f(a = 1, b = TRUE)
a == b
```

2.2 Quotes

```
# advised
"double quotes"
# discouraged
'single quotes'
```

In R, both double quotes and single quotes are available. The advantage of single quotes is that they result in a slightly cleaner visual representation of the code.

```
print('this is nice'); print("this is a bit less clean")
```

On the other hand, double quotes can be used to enclose single quotes, whereas the reverse is not possible. This can be useful to create a string like the following:

```r
varname <- "index"
found <- sample(c("n't ", ""), size = 1)
print(paste("the variable '", varname, "' was", found, " found", sep = ""))
```

```
## [1] "the variable 'index' was found"
```

Note that in this example, single quotes behave just like any other character. Hence, is not necessary to *close* an open single quote. This allows us to also use them as contraction in `varname` without escape. Depending on whether or not compatibility with other programming languages is required, double quotes are probably also a safer option in certain cases.

For these reasons, we advise the use of double quotes, so all strings are wrapped in the same type of quotes.

Chapter 3

Workflow & collaboration

Chapter 4

Package development