2. Let's consider using R

2.1 R has a lot going for it

R is a general tool (R Core Team (2022)). It is a statistical programming language (Ihaka & Gentleman (1996)). There are a lot of people using R. There are a lot of good reasons for this.

R is freely available

You can download and use R for free. What's not to like about that?

R can be used on any platform

It doesn't matter whether you are working on a PC, a Mac, or using a Linux operating system, R code works on any of these platforms, and R code is transferrable and can be shared.

R is open source

Because R is completely transparent and open source, there is a burgeoning global community of contributors. Anyone can write R code and share it openly with others. Anyone can make R packages or libraries and offer them to others. There are also a lot of free online groups/networks to support people in their quest to create R code for particular purposes and applications.

R code is useful for scripting/repeatability

Once you get your R code working to perform a specific analysis (and you are sure it does what you want it to do), let's suppose you now want to repeat that analysis hundreds of times. Because R is a programming language, it readily permits a straighforward avenue for scripting and repeatability.

R is always evolving and improving

The R community is always growing. Thus, both the R base package and contributed packages/libraries tend to continuously evolve and get better over time.

R is a language, so it is broad in scope

Because R is a language (rather than being a 'point-and-click' type of software), it is amenable to being used in lots of different ways by a lot of different communities. Everyone can shape (and share) their R code for their own needs. Indeed, you can find R packages and libraries implementing a very broad range of methods, which collectively services virtually any (perhaps all?) branches of statistics.

In short...

The above is not intended to be en exhaustive list of what is good about R, but it makes it easy to understand what makes R a useful tool. In short, I am a fan of R. I have used it in my teaching, and I use it a lot in my own statistical research, particularly for programming new statistical methods from scratch and testing them to see how they perform under different scenarios.

However, R is not the only thing I use, and there are certainly also some down-sides to using R. Let's consider some of those.

2.2 R has some down sides

Like any software, R has some down sides.

R has a steep learning curve

R is a programming language. It was invented by (and is used primarily by) statisticians. To use it successfully, you really do have to be comfortable writing and executing command-line code. So R is especially great if you are a statistician who is savvy in computer programming (or a programmer who likes statistics). R is great for doing statistical research, but it is not necessarily great for everyone.

It is not really appropriate to use R by just 'cutting and pasting' some R code that you find in someone else's examples (purporting to do what you wish to achieve) and merging it with your own R code for data analysis if you don't really know what those R commands actually do, nor what their assumptions are.

R can be really frustrating when your code doesn’t work, and you don’t know why.

It is also (unfortunately) very easy to make a mistake without even knowing it. The code may run, but is it doing what you think it is?

(Someone very knowledgeable in programming once teased me for being completely over the moon when my first bit of Fortran code would actually successfully compile without giving any errors. They simply smiled and said: "Ah, yes, but you don't really know if it **works** yet. Now you have to embark on all of the **testing** and **de-bugging**!" Sigh.)

To be sure about the R code you write (or even reasonably sure), you have to be comfortable digging in to the nuts and bolts of it. You have to know (or work out) how the R language works regarding different types of variables and objects. You have to know (or work out) what the assumptions are of every step you ask R to perform. You have to know (or work out) what any packages or dependencies you are using assume about the information you give it, and you have to know (or work out) what their limits of application are. This is not always (or even typically) a trivial task. All of this requires a reasonable amount of programming and debugging skills.

R packages vary in quality

With so many contributors, there is (necessarily) a great deal of variation in the quality of the available R code and R packages out there. Depending on who is making the contribution, there are different levels of programming sophistication lying 'under the hood'. Varying quality means available code has a wide range of reliability, particularly when used in new contexts.

In addition, every piece of R code varies in the level of available documentation, user notes and/or vignettes that accompany it. These are the things that will help you understand the underlying method, teach you how to use the package correctly, and identify what the assumptions and usage limits really are for that package. In some cases, the available information can be quite brief, sketchy, or cryptically written.

The extent and utility of 'warnings' and 'error' messages also varies greatly for different R packages. This is important, because such messages should help you to see where a problem is or highlight important limits, in the event that you run a given package and it doesn't work, or it runs into some sort of issue. Without good error messaging, you may never know that you are using a package outside the bounds of its intended use.

Given all of this, you should carefully and independently check any code or package that you intend to use to ensure its validity for your case. This sort of activity can be time-consuming and also prone to error unless you’ve got patience and good programming skills.

R package dependencies vary over time

Most R packages of reasonable complexity depend on several other R packages. Depending on the contributor and their level of commitment to the R package they have created, they may improve and update their package quite frequently or hardly ever. Of course, different packages are not necessarily updated by their individual authors at the same time.

This has a few consequences. First, it means that it can be quite challenging to keep all of the packages you want to use (and all of their dependencies) up to date.

Second (and even more annoying), code that worked just fine yesterday may not work today. Perhaps one of the packages that your code depended on has changed in the way it needs to be used, or in the naming conventions it deploys, etc. So even though one of the best things about R is the fact that everyone can contribute, it is also one of the most challenging things about it.

Although people like to imagine that R scripts are super great because they permit 'repeatability', the fact that R packages and their dependencies are in a constant state of flux means that R scripts, in fact, are not necessarily repeatable.

If code that used to work suddenly stops working, it is not always clear where the problem lies and (once again) de-bugging/programming skills are required. Even worse is the situation where underlying assumptions or defaults for a given package have changed. The author of the package might have great reasons for changing the defaults, but the result for you may be that your older code will run, but it will give you different results, and you won't know why. You will (once again) have to do some digging to figure it all out.

R makes assumptions 'under the hood'

R is a type of 'high-level' language. As such, it doesn't require you to declare the nature of your variables at the outset; such declarations are, for the most part, implicit (unless of course you choose to make them explicit). R therefore (necessarily) makes some assumptions about how to treat what you give it in any given context. For example, suppose you give R the following:

```
Factor.A <- c(1,1,1,2,2,2,3,3,3)
```

You might think you are giving R a factor, but it thinks you are giving it a vector of numbers. R will carry on regardless (it won’t necessarily give you an error), and you may be none the wiser.

There a lot of things like this that R will assume on the fly (some of which may be buried inside a package you choose to use), and unless you are knowledgeable about what these assumptions are, you can (all too easily) run your R code and get incorrect results. This is yet another reason why it is not wise to grab R code off the web and use it on your data without doing your own checks.

2.3 Pros and Cons of using R

To re-cap and summarise, below is a table outlining the primary pros and cons of using R, as I see it:

Pros	Cons
$\bullet$ A flexible programming language	$\bullet$ Steep learning curve
$\bullet$ Free, platform-independent	$\bullet$ Packages vary in quality and vary over time
$\bullet$ Open source	$\bullet$ Updating can be tricky
$\bullet$ Lots of contributors	$\bullet$ Assumptions can be cryptic
$\bullet$ Broad in scope	$\bullet$ Frustrating when code won't run
$\bullet$ Always evolving	$\bullet$ When it runs... is it correct?
$\bullet$ Great for stats research	$\bullet$ Requires independent checks/debugging/programming skills