**Introduction**

Let’s say you want to write a function with optional functionality which is dependent on the installation of a package that your colleague may not have installed. For example, let’s say you want to have an option to return a data.table (or a tibble) instead of a data.frame, but in this case you don’t want to force your function’s user to have to install data.table (or tibble – and its dependencies) just to use your function. Maybe they can’t install it because they are restricted to do so by their IT department or maybe they are working offline. Is it possible to do this?

**A Toy Example**

Let’s say we have a simple function which takes a data.frame and adds a new column which is a multiplication of an existing column, before returning the whole data.frame with that new column.

toy\_function <- function(data, column, multiple = 2L, as\_data\_table = FALSE) {

stopifnot(is.integer(multiple) || is.numeric(multiple))

new\_column\_name <- paste(column, multiple, sep = "\_")

data[, new\_column\_name] <- data[, column] \* multiple

if (as\_data\_table) data <- data.table::setDT(data)

return(data)

}

Running this function with as\_data\_table = TRUE without data.table installed will give the following error:

toy\_function(mtcars, "mpg", as\_data\_table = TRUE)

# Error in loadNamespace(name) : there is no package called ‘data.table’

This is a frustration for the user. This also means that the whole function no longer works and doesn’t return anything. So what can we do? Well, this is where the function requireNamespace() comes in handy.

requireNamespace is a wrapper for loadNamespace analogous to require that returns a logical value.

Using requireNamespace(), we can test whether or not the data.table package can be loaded from the user’s library before running certain functionality. Let’s take a look at how this changes our function:

toy\_function <- function(data, column, multiple = 2L, as\_data\_table = FALSE) {

stopifnot(is.integer(multiple) || is.numeric(multiple))

new\_column\_name <- paste(column, multiple, sep = "\_")

data[, new\_column\_name] <- data[, column] \* multiple

if (as\_data\_table) {

if (!requireNamespace("data.table", quietly = TRUE)) {

warning("Please install package 'data.table' when using 'as\_data\_table = TRUE'")

return(data)

}

data <- data.table::setDT(data)

}

return(data)

}

Now when we run our function, the function will check for a data.table installation and if it is not available, it will warn us that we need to install data.table in order to use this functionality; yet it will still return the manipulated data, just as a data.frame.

toy\_function(mtcars, "mpg", as\_data\_table = TRUE)

# mpg cyl disp hp drat wt qsec vs am gear carb mpg\_2

# Mazda RX4 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4 42.0

# Mazda RX4 Wag 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4 42.0

# Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1 45.6

# ...

# Ferrari Dino 19.7 6 145.0 175 3.62 2.770 15.50 0 1 5 6 39.4

# Maserati Bora 15.0 8 301.0 335 3.54 3.570 14.60 0 1 5 8 30.0

# Volvo 142E 21.4 4 121.0 109 4.11 2.780 18.60 1 1 4 2 42.8

# Warning message:

# In toy\_function(mtcars, "mpg", as\_data\_table = TRUE) :

# Please install package 'data.table' when using 'as\_data\_table = TRUE'

A real example of this can be seen in the [fst](https://github.com/fstpackage/fst) package. When using the fst::read\_fst() function, the user has the option to return their loaded data [as a data.table](https://github.com/fstpackage/fst/blob/develop/R/fst.R#L172).

I really like how this way of using optional functionality does not force additional package downloads on people and also means that your code remains usable on restricted servers or offline. It’s also a great way to not clog up people’s libraries (I’m looking at you tidyverse).

**Package Development**

This solution extends further when developing an R package. My current team is due to start utilising Spark, though our Spark cluster is not yet configured. I have therefore been testing new functionality using a local Spark cluster on our dev environment. Our production environment does not have the sparklyr package installed yet and so I cannot include any sparklyr code within my codebase…or can I?

Typically when your package relies on another package for functionality, you list that package as an Import within your package’s DESCRIPTION file. But what this typically means is that when someone installs your package, they will also need to install the Imports. However if the dependency is not available to install, you will receive the following error upon installation.

==> R CMD INSTALL --no-multiarch --with-keep.source mypackage

\* installing to library ‘/Library/Frameworks/R.framework/Versions/3.5/Resources/library’

ERROR: dependency ‘sparklyr’ is not available for package ‘mypackage’

\* removing ‘/Library/Frameworks/R.framework/Versions/3.5/Resources/library/mypackage’

Exited with status 1.

I am forgoing the idea that it probably isn’t best practice to include non-production code which will not work in your package, this blog post is for the purposes of demonstration only; the following is what you *could* do.

Removing sparklyr from the Import list would allow us to install the package but we would then face two new issues. Firstly, end users could potentially run the function (even if it isn’t exported) and be faced with that same unhelpful error message we saw earlier.

my\_function()

# Error in loadNamespace(name) : there is no package called ‘sparklyr’

Secondly, from a developer’s point of view, the R CMD check would fail – which would in turn fail any CI/CD pipelines.

devtools::check()

# ...

# ❯ checking dependencies in R code ... WARNING

# '::' or ':::' import not declared from: ‘sparklyr’

So within my function, I simply place the below code (or similar) and if the user tries to run the function, it will simply stop and tell them they need to install the sparklyr package.

if (!requireNamespace("sparklyr", quietly = TRUE)) {

stop("Package sparklyr needed.")

}

This will subsequently pass the R CMD check. Were this an open source package, I could add sparklyr to the Suggests field of the DESCRIPTION file such that users could install the sparklyr package to get the additional functionality if they wanted it (this is exactly what the fst package does).

**Conclusion**

So to conclude if you want to include functionality in your code which relies on other packages but are worried about people not having access to those packages, or simply don’t want to force your users to have to install the additional packages, then consider requireNamespace(). This is a great way of offering additional functionality without the need to clog up user’s libraries.