At [Jumping Rivers](https://www.jumpingrivers.com/) we run a lot of R courses. Some of our most popular courses revolve around the [tidyverse](https://www.tidyverse.org/), in particular, our [Introduction to the tidyverse](https://www.jumpingrivers.com/training/course/introduction-r-tidyverse/) and our more advanced [mastering](https://www.jumpingrivers.com/training/course/r-tidyverse-purrr-broom/) course. We even trained over 200 data scientists NHS – see our [case study](https://www.jumpingrivers.com/case-study/project-1-training/) for more details.

As you can imagine, when giving an on-site course, a reasonable question is what version of R is required for the course. We **always** have an RStudio cloud back-up, but it’s nice for participants to run code on their own laptop. If participants are to bring there own laptop it’s trivial for them to update R. But many of our clients are financial institutions or government where an upgrade is a non-trivial process.

So, what version of R is required for a *tidyverse* course? For the purposes of this blog post, we will define the list of packages we are interested in as

library("tidyverse")

tidy\_pkgs = c("ggplot2", "purrr", "tibble", "dplyr", "tidyr", "stringr",

"readr", "forcats")

The code below will work with **any** packages of interest. In fact, you can set pkgs to all R packages in CRAN, it just takes a while.

**Package descriptions**

In R, there is a handy function called available.packages() that returns a matrix of details corresponding to packages currently available at one or more repositories. Unfortunately, the format isn’t initially amenable to manipulation. For example, consider the **readr** package

readr\_desc = available.packages() %>%

as\_tibble() %>%

filter(Package == "readr")

I immediately converted the data to a tibble, as that

* changed the rownames to a proper column
* changed the matrix to a data frame/tibble, which made selecting easier

Looking at the read\_desc, we see that it has a minimum R version

readr\_desc$Depends

## [1] "R (>= 3.0.2)"

but due to the format, it would be difficult to compare to R versions. Also, the list of imports

readr\_desc$Imports

## [1] "Rcpp (>= 0.12.0.5), tibble, hms, R6"

has a similar problem. For example, with the data in this format, it would be difficult to select packages that depend on **tibble**.

**Tidy package descriptions**

We currently have four columns

* Imports, Depends, Suggests, Enhances

each entry in these columns contains multiple packages, with possible version numbers. To tidy the data set I’m going to create four new columns:

* depend\_type: one of Imports, Depends, Suggests, Enhances and LinkingTo
* depend\_package: the package name
* depend\_version: the package version
* depend\_condition: something like equal to, less than or greater than

The hard work is done by the function clean\_dependencies(), which is at the end of the blog post. It essentially just does a bit of string manipulation to separate out the columns. The function works per package, so we iterate over packages using map\_df()

pkg\_deps = c("Depends", "Enhances", "Suggests", "Imports", "LinkingTo")

av\_pkgs = available.packages() %>%

as\_tibble()

av\_pkgs = map\_df(pkg\_deps, clean\_dependencies, av\_pkgs = av\_pkgs) %>%

select(-pkg\_deps)

# Std version: 3.4 -> 3.4.0

av\_pkgs = av\_pkgs %>%

mutate(depend\_version = if\_else(depend\_package == "R",

clean\_r\_ver(depend\_version),

depend\_version))

# Standardise column names

colnames(av\_pkgs) = str\_to\_lower(colnames(av\_pkgs))

After this step, we now have tibble with tidy columns:

av\_pkgs %>%

select(package, depend\_type, depend\_package, depend\_version, depend\_condition) %>%

slice(1:4)

## # A tibble: 4 x 5

## package depend\_type depend\_package depend\_version depend\_condition

##

## 1 A3 Depends R 2.15.0 >=

## 2 abbyyR Depends R 3.2.0 >=

## 3 abc Depends R 2.10.0 >=

## 4 abc.data Depends R 2.10.0 >=

and we can see minimum R version the package authors have indicated for their package. **However**, this isn’t the minimum version required. Each package imports a number of other packages, e.g. the **readr** imports 4 packages

av\_pkgs %>%

filter(package == "readr" & depend\_type == "Imports") %>%

select(depend\_package) %>%

pull()

## [1] "Rcpp" "tibble" "hms" "R6"

and each of those packages, also import other packages. Clearly, the minimum version required to install **dplyr** is the maximum R version of all imported packages

**Some interesting things**

Before we work out the maximum R version for each set of imports, we should first investigate how many imports each package using a bit of **dplyr**

imp = av\_pkgs %>%

group\_by(package, depend\_type) %>%

summarise(n = length(depend\_package)) %>%

arrange(package, n) %>%

filter(depend\_type != "Enhances" & depend\_type != "LinkingTo")

imp %>%

group\_by(depend\_type) %>%

summarise(mean(n))

## # A tibble: 3 x 2

## depend\_type `mean(n)`

##

## 1 Depends 1.96

## 2 Imports 4.98

## 3 Suggests 3.63

Using histograms we get a better idea of the overall numbers. Note that here we’re using the ipsum theme from the **hrbrthemes** package.

library(hrbrthemes)

ggplot(imp, aes(n)) +

geom\_histogram(binwidth = 1, colour= "white", fill = "steelblue") +

facet\_wrap(~depend\_type) +

xlim(c(0, 25)) + ylim(c(0, 6000)) +

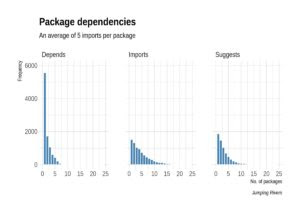
theme\_ipsum() +

labs(x="No. of packages", y="Frequency",

title="Package dependencies",

subtitle="An average of 5 imports per package",

caption="Jumping Rivers")



**Maximum overall imports**

As I’ve mentioned, we need to obtain not just the imported packages, but also their dependencies. Fortunately, the **tools** package comes to our rescue,

tools::package\_dependencies("readr",

which = c("Depends", "Imports", "LinkingTo"),

recursive = TRUE) %>%

unlist() %>%

unname()

## [1] "Rcpp" "tibble" "hms" "R6" "BH"

## [6] "methods" "pkgconfig" "rlang" "utils" "cli"

## [11] "crayon" "pillar" "assertthat" "grDevices" "fansi"

## [16] "utf8" "tools"

Using the package\_dependencies() function, we simply

* Obtain a list of dependencies for a given package
* Extract the maximum version of R for all packages in the list

At the end of this post, there are two helper functions:

* max\_r\_version() – takes a vector of R versions, and returns a maximum version. E.g.
* max\_r\_version(c("3.2.0", "3.3.2", "3.2.0"))
* ## 3.3.2
* get\_r\_ver() – calls package\_dependencies() and returns the maximum R version out of all of the dependencies.

Also, we have simplified some the details and what we’ve done isn’t quite right – it’s more of a first approximation. See the end of the post for details

Now that’s done, we can pass the list of tidyverse packages then compare their stated R version, with the actual required R version

# Here we are using just the tidyverse

# But this works with any packages

tidy = map\_dfr(tidy\_pkgs, get\_r\_ver, av\_pkgs) %>%

group\_by(package) %>%

slice(1) %>%

select(package, r\_real, r\_cur)

We then select the packages where there is a difference

tidy %>%

filter(r\_real != r\_cur | (!is.na(r\_real) & is.na(r\_cur)))

# A tibble: 3 x 3

# Groups: package [3]

package r\_real r\_cur

1 readr 3.1.0 3.0.2

2 tidyr 3.1.2 3.1.0

The largest difference in R versions is for **readr** (which feeds into the **tidyverse**). **readr** claims to only need R version 3.0.2 but a bit more investigation shows that **readr** depends on the tibble package which is version 3.1.0. **Although**, it is worth noting that 3.1.0 is fairly old!

**Take away lessons**

The takeaway message is that dependencies matter. A single change affects everything in the package dependency tree. The other lesson is that the tidyverse team have been very **careful** about there dependencies. In fact, all of their packages are checked on R 3.1, 3.2, …, devel

**Simplifications: skipping package versions**

In this analysis, we’ve completely ignored version numbers and always assumed we need the latest version of a package. This clearly isn’t correct. So to do this analysis properly, we would need the historical DESCRIPTION files for packages and use that to determine versions.

Thanks to Jim Hester who spotted an error in a previous version of this post.

**Functions**

clean\_dependencies = function(av\_pkgs, type) {

no\_of\_cols = max(str\_count(av\_pkgs[[type]], ","), na.rm = TRUE) + 1

cols = paste0(type, seq\_len(no\_of\_cols))

av\_clean = av\_pkgs %>%

separate(type, sep = ",", into = cols, fill = "right", remove = FALSE) %>%

gather(key = type, pkg, cols) %>%

drop\_na(pkg) %>%

mutate(type = !!type) %>%

separate(pkg, sep = "\\(", into = c("package", "version"), fill = "right") %>%

mutate(version = str\_remove(version, "\\)"),

package = str\_remove(package, "\\)")) %>%

mutate(version = str\_remove(version, "\n"),

package = str\_remove(package, "\n")) %>%

mutate(version = str\_trim(version),

package = str\_trim(package)) %>%

filter(package != "")

## Detect where version is

version\_loc = str\_locate(av\_clean$version, "[0-9]")

av\_clean %>%

mutate(condition = str\_sub(version, end = version\_loc[, "end"] - 1),

version = str\_sub(version, start = version\_loc[, "end"])) %>%

rename(depend\_type = type,

depend\_package = package,

depend\_version = version,

depend\_condition = condition) %>%

mutate(depend\_condition = str\_trim(depend\_condition))

}

clean\_r\_ver = function(vers) {

nas = is.na(vers)

vers[!nas] = paste0(vers[!nas], if\_else(str\_count(vers[!nas], "\\.") == 2, "", ".0"))

vers

}

max\_r\_version = function(r\_versions) {

s = str\_split(r\_versions, "\\.", n = 3)

major = map(s, `[`, 1) %>%

map(as.integer)

if(sum(major == 3) > 0) {

maj\_number = 3

} else if(sum(major == 2) > 0) {

maj\_number = 2

} else if(sum(major == 1) > 0) {

maj\_number = 1

} else {

return("")

}

minor = map(s, `[`, 2) %>%

map(as.integer)

min\_number= minor[which(major == maj\_number)] %>%

unlist() %>%

max()

third = map(s, `[`, 3) %>%

map(as.integer)

third\_number = third[which(major == maj\_number & minor == min\_number)] %>%

unlist() %>%

max()

glue::glue("{maj\_number}.{min\_number}.{third\_number}")

}

get\_r\_ver = function(pkg, clean\_av) {

r\_cur = clean\_av %>%

filter(depend\_package == "R" & package == pkg) %>%

select(depend\_version) %>%

pull()

if(length(r\_cur) == 0) r\_cur = NA

pkg\_dep = tools::package\_dependencies(pkg,

which = c("Depends", "Imports", "LinkingTo"),

recursive = TRUE) %>%

unlist() %>%

unname() %>%

c(pkg)

# Optional and recommended packages

with\_r = filter(av\_pkgs, !is.na(priority)) %>%

select(package) %>% pull() %>%

unique()

pkg\_dep = pkg\_dep[!(pkg\_dep %in% with\_r)]

r\_ver = clean\_av %>%

filter(package %in% pkg\_dep) %>%

filter(depend\_type == "Depends") %>%

filter(depend\_package == "R") %>%

select(depend\_version) %>%

summarise(max\_r\_version(depend\_version)) %>%

pull()

clean\_av %>%

filter(package == pkg) %>%

mutate(r\_real = as.character(r\_ver),

r\_cur = r\_cur)

}