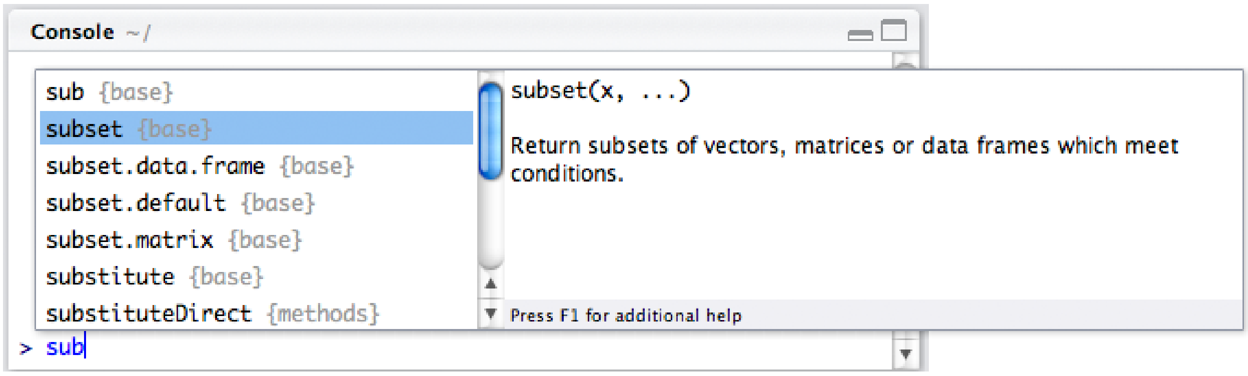
Packages in R

Author: Cole Brookson **Date:** 22 June 2022

What Are Packages?

R packages are collections of related functions, code, and data, which can be downloaded and used by any R user, as a way of avoiding having to write all the code to do a task manually themselves. All official packages are submitted and verified through CRAN (Comprehensive R Archive Network). While a number of functions and capabilities ship natively with "base" R, packages typically are sets of functions that perform related tasks (e.g. data management). For example, if in an R script or `.Rmd` file, we begin to type the function `subset()`, we get the following:


```
Console ~/
sub {base}
subset {base}
subset.data.frame {base}
subset.default {base}
subset.matrix {base}
substitute {base}
substituteDirect {methods}
> sub
```

```
subset(x, ...)

Return subsets of vectors, matrices or data frames which meet conditions.

Press F1 for additional help
```

We see here that R is suggesting some options to us of functions that start with `sub`. The word in the curly braces, `{base}` indicates that that function is native to "base" R, or the base set of functions and methods that we have access to without loading in other packages. When you download R, a set of packages are imported automatically, including `base`, `stats`, and some others. Packages not downloaded automatically with R need to be explicitly loaded separately.

There are many many packages, the vast majority of which we will not even talk about. In fact, as of November 2020, more than 16,000 packages were available on CRAN (and that's not including the non-CRAN packages that are available too!).

Note that many packages are termed *dependencies*, that is, they are not the packages we are going to use directly, but they are required on our computers, for the package we **do** want to use to work properly.

Loading Packages

We can check what packages we have installed on our computer by running the `library()` command.

```r
library()
```

```
## Warning in library(): libraries '/usr/local/lib/R/site-library', '/usr/lib/R/
## site-library' contain no packages
```

On this computer, that returns a large number of packges, but on your computer it may return less.

The EEB R Manual will make use frequently of packages that are part of the *Tidyverse*. The Tidyverse is "...an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures" that is funded and developed by the RStudio foundation. See more about our opinions on the tidyverse elsewhere. One of the first packages we'll use is the `dplyr` package, a very common and useful data cleaning package.

As a demonstration, let's install the `dplyr` package:

```r
install.packages("dplyr")
```

```
## Installing package into '/home/cole/R/x86_64-pc-linux-gnu-library/4.2'
## (as 'lib' is unspecified)
```

You will get a long output as the package installs, and you can tell if it has successfully installed, by attempting to load the package via the `library()` function:

```
library("dplyr")
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

There is some important information here in this library load. First of all, we see that two packages, `package:stats` and `package:base` now have "masked objects". The objects that are masked for each of these packages are listed. For `stats`, `filter()` and `lag()` are masked, and for `base`, the functions `intersect()`, `setdiff()`, `setequal()`, and `union()` are masked.

Don't worry, you don't have to do anything about this, but just know that **this means there are functions in the `dplyr` package with the same names (i.e. `filter()` and `lag()`, and `intersect()`, `setdiff()`, `setequal()`, and `union()`) as functions that were already loaded in R. So, if you just call the function `filter()` it will default to using the function named `filter()` from the `dplyr` package, and NOT from the `stats` package.

It is good practice to be explicit about what package you are using for what task with the `package::function()` notation. That is, if I wanted to use the `filter()` function from `dplyr`, I would write: `dplyr::filter()`, but if I wanted to use the function from the `stats` package I would write `stats::filter()`. This way R will not get confused and perhaps default to the wrong package!

Using Packages in Scripts

As we will discuss in the "Workflow" section, any packages that we need for an analysis should always be loaded at the top of the script we're working in. For example, if I wanted to use two packages in a script, `here` and `dplyr`, I would load them both at the very top of my new R script like this:

```
library(here)
```

```
## here() starts at /home/cole/Github/eeb-r-manual
```

```
library(dplyr)
```

Note that I do **not** use the `install.packages()` function here, as once a package is downloaded once, we don't need to download it again, and it will save us time to not do so. It is usually recommended to run any installations in the console, or else to comment them out of our code like this:

```
# install.packages("here", "dplyr")
library(here)
library(dplyr)
```

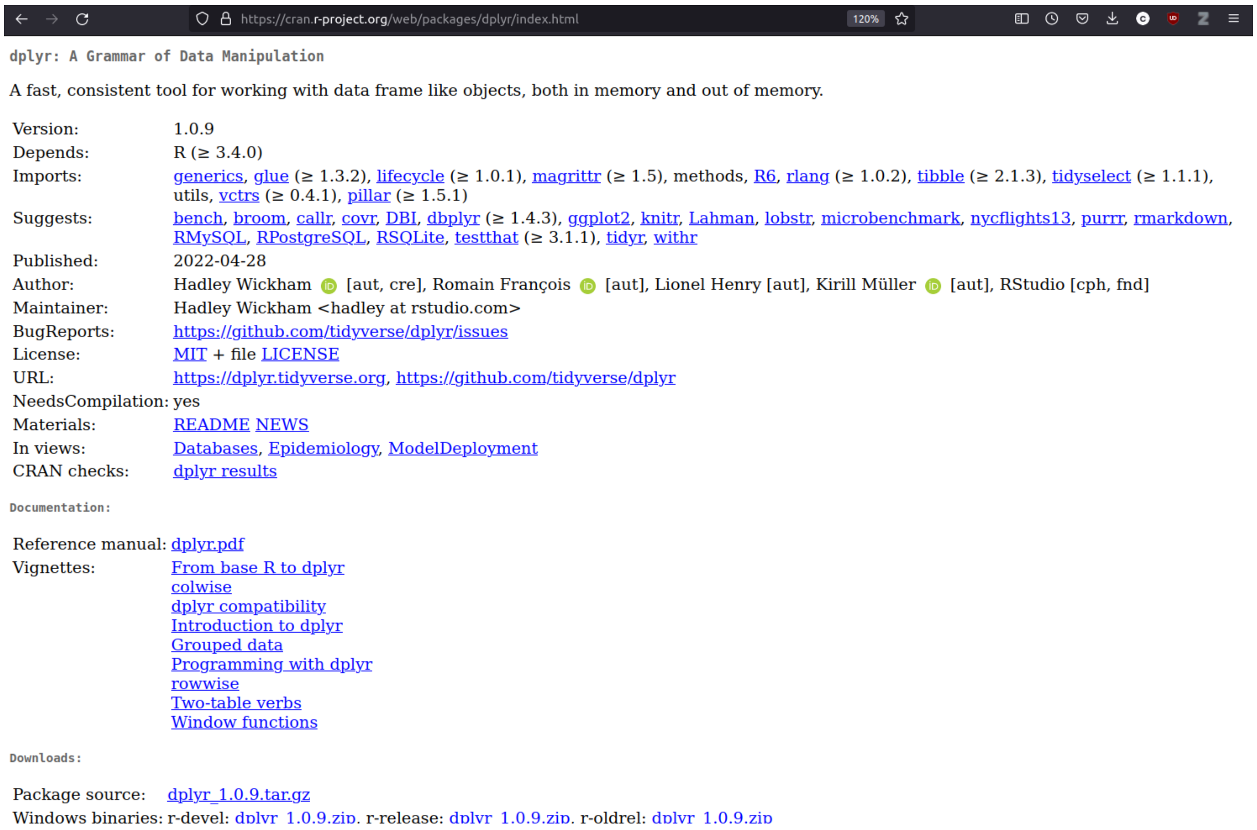
Learning More About a Package

As you begin to learn R, you'll often hear or read the phrase "there's a package for that!"

Almost any task you can imagine wanting to do in R probably has a package to achieve that task that someone has written before. While some packages (like the `Tidyverse` packages) are written by software developers who are paid to do so, most packages are "Open Source" which means that they are written by enthusiasts (who may also be software developers, but are not explicitly paid to write R packages) in their spare time, and are made openly available to anyone who wants to use them.

When you begin using a package, you may have questions about how it works. All packages on CRAN come with a "vignette", a worked example using some functions from the package to demonstrate it's usefulness. You can access these vignettes on CRAN TaskViews. The TaskViews are also helpful as they are organized by the type of Task you may want to do, and can help direct you to a package that may work for your purposes.

As an example, we can search for the **dplyr** package on the CRAN website, and navigate to it's CRAN homepage:



https://cran.r-project.org/web/packages/dplyr/index.html

120%

dplyr: A Grammar of Data Manipulation

A fast, consistent tool for working with data frame like objects, both in memory and out of memory.

Version:	1.0.9
Depends:	R (≥ 3.4.0)
Imports:	generics, glue (≥ 1.3.2), lifecycle (≥ 1.0.1), magrittr (≥ 1.5), methods, R6, rlang (≥ 1.0.2), tibble (≥ 2.1.3), tidyselect (≥ 1.1.1), utils, vctrs (≥ 0.4.1), pillar (≥ 1.5.1)
Suggests:	bench, broom, callr, covr, DBI, dbplyr (≥ 1.4.3), ggplot2, knitr, Lahman, lobstr, microbenchmark, nycflights13, purrr, rmarkdown, RMySQL, RPostgreSQL, RSQLite, testthat (≥ 3.1.1), tidyr, withr
Published:	2022-04-28
Author:	Hadley Wickham [aut, cre], Romain François [aut], Lionel Henry [aut], Kirill Müller [aut], RStudio [cph, fnd]
Maintainer:	Hadley Wickham <hadley at rstudio.com>
BugReports:	https://github.com/tidyverse/dplyr/issues
License:	MIT + file LICENSE
URL:	https://dplyr.tidyverse.org, https://github.com/tidyverse/dplyr
NeedsCompilation:	yes
Materials:	README NEWS
In views:	Databases, Epidemiology, ModelDeployment
CRAN checks:	dplyr results

Documentation:

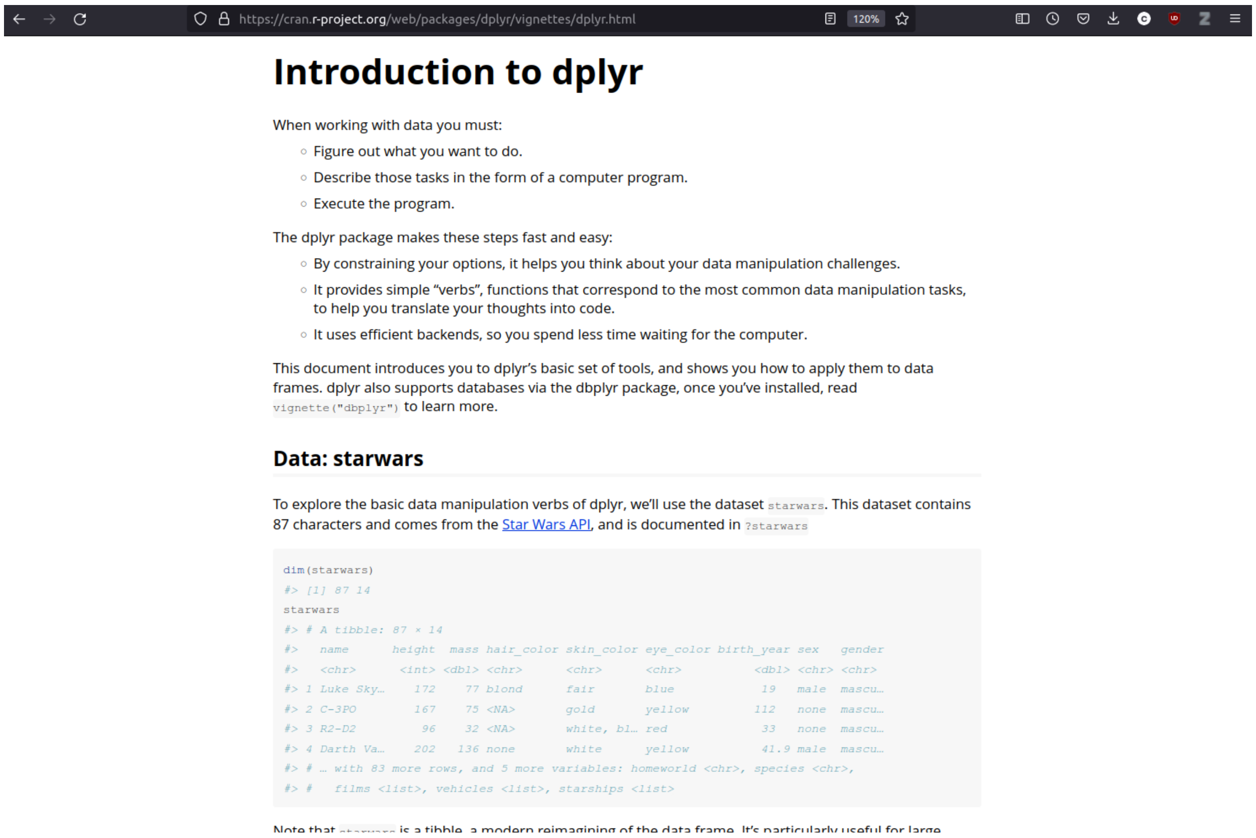
Reference manual:	dplyr.pdf
Vignettes:	From base R to dplyr colwise dplyr compatibility Introduction to dplyr Grouped data Programming with dplyr rowwise Two-table verbs Window functions

Downloads:

Package source:	dplyr_1.0.9.tar.gz
Windows binaries:	r-devel: dplyr_1.0.9.zip, r-release: dplyr_1.0.9.zip, r-oldrel: dplyr_1.0.9.zip

And we see there is a whole section for vignettes at the bottom. Since **dplyr** is a **Tidyverse** package, the documentation is extensive, significantly more than most other packages.

If we click on the *Introduction to dplyr* page, we'll find out lots about how to use **dplyr** in a convenient format:


https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html
120%

Introduction to dplyr

When working with data you must:

- Figure out what you want to do.
- Describe those tasks in the form of a computer program.
- Execute the program.

The dplyr package makes these steps fast and easy:

- By constraining your options, it helps you think about your data manipulation challenges.
- It provides simple "verbs", functions that correspond to the most common data manipulation tasks, to help you translate your thoughts into code.
- It uses efficient backends, so you spend less time waiting for the computer.

This document introduces you to dplyr's basic set of tools, and shows you how to apply them to data frames. dplyr also supports databases via the dbplyr package, once you've installed, read `vignette("dbplyr")` to learn more.

Data: starwars

To explore the basic data manipulation verbs of dplyr, we'll use the dataset `starwars`. This dataset contains 87 characters and comes from the Star Wars API, and is documented in `?starwars`

```
dim(starwars)
#> [1] 87 14
starwars
#> # A tibble: 87 × 14
#>   name      height  mass hair_color skin_color eye_color birth_year sex   gender
#>   <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
#> 1 Luke Sky…    172    77 blond      fair       blue            19   male  mascu…
#> 2 C-3PO        167    75 <NA>       gold       yellow         112   none  mascu…
#> 3 R2-D2         96    32 <NA>       white, bl… red             33   none  mascu…
#> 4 Darth Va…    202   136 none       white      yellow          41.9 male  mascu…
#> # … with 83 more rows, and 5 more variables: homeworld <chr>, species <chr>,
#> #   films <list>, vehicles <list>, starships <list>
```

Note that `starwars` is a tibble, a modern reimagining of the data frame. It's particularly useful for large