UPPSALA
UNIVERSITET

Uppsala Child & Baby Lab

Packages, Package management, and the Tidyverse

R for Psychology Research

Marcus Lindskog, Docent
2019-09-08

Overview

1. Packages
2. Package management
3. The `Tidyverse`

Packages

What is a package?

1. A collection of **functions**, **data**, and **documentation**.
2. Extends the capabilities of base R.
3. Written by contributers to solve one or more statistical problem.
4. Available from CRAN or from the developer (e.g., GitHub).

How do I find a useful package?

1. Google what it is you want to do!
2. Search https://Rdocumentation.org.
3. Search MRAN (the Microsoft R Application Network) https://mran.microsoft.com/packages
4. See the Trending R repositories list on GitHub.
5. See CRAN Tak views https://cran.r-project.org/web/views/

How do I know which to use?

- Many packages do the same thing.
- How do I pick one that best does the job?

1. Try it out.
2. Ask friends.
3. Ask on social media, like twitter. The R-community is **very** helpful.

Package management

Install a package.

- To install a package use:

```
install.packages("packagename")

#For example

install.packages("nlme")
```

- To install the `nlm` packages that does mixed linear models.
- To check which packages that are currently installed.

```
installed.packages()
```

Load a packages

- Before you can use any of the functions or data in a package you need to make it available to your R session.
- This is done by loading your library.
- You can load your library from a script using:

```r
library(nlme)
```

- This is the recomended way, but R studio also allows you to load packages from the `Packages` tab.

Load packages

- It is good practice to begin your analysis scripts by loading all packages that are needed for your analysis.

```r
#Load packages
library(plyr)
library(tidyverse)
library(knitr)
library(lubridate)
library(lme4)
```

Detach

- If you, for some reason, want to remove a loaded library from your session, you need to detach it.

```r
library(BayesFactor)
detach("package:BayesFactor", unload = TRUE)
```

Documentation

- If you want to know what a packages does, read the documentation.
- This can be found on https://Rdocumentation.org
- And on https://cran.r-project.org
- The documentation often includes **Vignettes** and **Examples**.

Updating

- Updating packages can sometimes be a hazzel.
- You should know that updating can change functionality.
- When you update your R version, all packges needs to be reinstalled. On Windows, this can be done using the `installr` package.
- On Mac, take some time to google a workflow.
- If you only want to check that all your packages are up to date, use:

```
update.packages()
```

Tidyverse

Tidyverse

- Preparing, wrangling, and visualizing data is a large part of doing statistics.
- There are many functions and packages in R to help you with that.
- However, the `Tidyverse` provides a collection of packages that work very well toghether and that are developed to solve these specific tasks.

Tidy data

- The `Tidyverse` is developed by Hadly Wickham and builds on his idea of of **tidy data**.
- Tidy data

  1. Each variable forms a column.
  2. Each observation forms a row.
  3. Each type of observational unit forms a table.

- This is acctually very close to how we often think of a data set in psychology.
- The `Tidyverse` therefor works very well for our types of data.

`Tidyverse` overview

- We will work with many of the packages from the `Tidyverse` later in the course.
- Here I will only give you a brief overview and some examples.

`Tidyverse` packages

- The `Tidyverse` includes the following packages.
- Data Wrangling and Transformation
  - dplyr
  - tidyr
  - stringr
  - forcats
- Data Import and Management
  - tibble
  - readr
- Functional Programming
  - purrr
- Data Visualization and Exploration
  - ggplot2

dplyr

- A very simple and agile package for data manipulation.
- Can use the pipe operator `%>%` to combine functions. This is very useful, as we will see later.
- Examples of functions from `dplyr`:
    - `select()`: Select columns from your dataset
    - `filter()`: Filter out certain rows that meet your criteria(s)
    - `group_by()`: Group different observations together.
    - `summarise()`: Summarise any of the above functions
    - `arrange()`: Arrange your column data
    - `join()`: Perform left, right, full, and inner joins in R
    - `mutate()`: Create new columns by preserving the existing variables

tidyr

- Complements the `dplyr`packages by providing functinos to arrange columns.
- Examples of functions from `tidyr`
    - `gather()`: Gathers multiple columns and converts them into key-value pairs.
    - `spread()`: Takes two columns and spreads them into multiple columns.
    - `separate()`: Helps in separating or splitting a single column into numerous columns
    - `unite()`: Works opposite to the separate() function. Combines two or more columns into one

stringr

- Working with strings often gives you a headache.

--

- stringer makes it a lot easier.

--

- Examples of functions from `stringr`
    - `str_sub()`: Extract substrings from a character vector
    - `str_trim()`: Trim white spaces
    - `str_remove()`: Removes a pattern from a character vector.
    - `str_to_lower/str_to_upper`: Converts into lower case or upper case

Example

```
od_data <- read_csv('analyses/data/OD_results.txt') %>%
  filter(!event_name == "-") %>%
  select(name, trialNr, event_name, AOI_name, AOI_value) %>%
  rename(ID = name) %>% rename(trial = trialNr) %>%
  mutate_at("event_name", str_remove, pattern = "a") %>%
  mutate(trial = as.numeric(trial),
         event_name = as.factor(event_name),
         AOI_value = as.numeric(AOI_value)) %>%
  spread(AOI_name, AOI_value) %>%
  mutate_at("ID", str_remove, pattern = ".csv") %>%
  separate(ID, c("ID", "R1", "Session", "Date",
                 "Time","X1","X2", "X3"), sep = "_") %>%
  select(-c("R1", "X1","X2", "X3")) %>%
  mutate_at("ID", str_remove, pattern = "S") %>%
  mutate(ID = as.numeric(ID),
         Session = as.factor(Session),
         Date = ymd(Date),
          PS = ifelse(Left_AOI < .200 | Right_AOI < .200  |
                        screen < 12.5*.25, NA, PS))
```

readr

- Getting your data into R is sometimes a tricky task.
- The `readr` package provides functions that are very efficient to read in (and write) flat files.

- Examples of functions in `readr`

    - `read_delim()`: reads delimited files.
    - `read_csv()`: reads csv-files.
    - `write_csv()`: writes csv-files.

- You might also need to get data into R that come proprietary file formats (e.g., SAS, SPSS, Excel).

--

- To help you with this task we use the packages `haven` and `readxl`.

tibble

- The `tibble` packages introduces a new data structure into R, the **tibble**.
- A tibble is a modern take on the data frame.
- It makes it easier and more consistent to work with tidy data in R.

ggplot2

- R is perhaps best known for the graphs it can produce.
- `ggplot2` is one of the moste versitaile and powerful packages for graphing in R.
- It builds on Edward Tufte's **grammar of graphics**.
- Using a few basic elements, it can build nice looking graphs.

Example

```
library(tidyverse)

play_data <- cars
median_speed <- median(play_data$speed)
median_dist <- median(play_data$dist)
play_data$fast_long <- factor(ifelse(play_data$speed > median_speed & pla
                              labels = c("Shit", "Super"))
```

Example

```
my_plot <- ggplot(play_data, aes(x = speed, y = dist,
                                 color =  fast_long)) +
  geom_point()+
  labs(title = "Cars data",
       subtitle = "An investigation",
       x = "Speed",
       y = "Distance",
       color = "Type of car") +
  theme_minimal() +
  scale_color_manual(values = c("grey", "red"))+
  theme(panel.grid = element_blank(),
        axis.line = element_line(color = "grey"),
        legend.position = c(.10,.85)) +
  geom_smooth(method="lm", color = "grey",
              linetype = "dashed", se = FALSE) +
  geom_hline(yintercept = median_dist,
             color = "grey", linetype = "dashed") +
  geom_vline(xintercept = median_speed,
             color = "grey", linetype = "dashed") +
  annotate("text", label = "Median Distance",
           x = 5, y = median_dist + 4, size = 4,
           colour = "grey")+
  annotate("text", label = "Median Speed",
           x = median_speed + 1.5, y = 120,
```

That's all folks!