Start with RMarkdown

A gentle slide (show) into R-package development

Marc A.T. Teunis

2019-12-04 11:01:48

Contents

Prerequisites

- Windows: Install Rtools
- Clone Materials from:
  https://github.com/UtrechtUniversity/R-data-cafe/start_with_rmd
- Install packages:

```
library(tidyverse)
library(here)
library(usethis)
library(devtools)
library(reprex)
```

Intro

Discuss with your neighbour (2 min.)

1. When was the last time you wrote a function in R?
2. What did it do?

R Markdown

- An 'R' implementation of the simple Markdown mark-up language
- Combines prose with R code and output
- Many output formats
- Accepts html, css, LaTeX
- Is generally a good starting point when setting up an analysis
- As it turns out: it is also a good starting point for creating an R-package

Why start with RMarkdown (for an R-package)

1. RMarkdown enforces more elaborate documentation (narratives + code + output)
2. Analysis follows a natural flow
3. Functions are put into context of application
4. Information, analysis and functions are collected in one place
5. When creating a package, you already have a vignette (documentation)

Why build an R-package

1. R-package is the natural 'communication' vessel for R-code
2. Documentation increases reproducibility (future you will be grateful)
3. R-packages are the natural habitat for R functions
4. Do not repeat yourself (code + data + documentation all in one place without redundancy)

Imagine this data

```
data_dengue <- read_csv(
  here::here(
    "data-raw", "dengue_data.csv"),
  comment = "#")

data_dengue[c(1:5), c(7:11)]
```

```
## # A tibble: 5 x 5
##   Mexico Philippines Singapore Thailand Venezuela
##    <dbl>       <dbl>     <dbl>    <dbl> <chr>
## 1 NA            999     0.059       NA missing
## 2 NA            999     0.059       NA missing
## 3  0.071        999     0.238       NA mising
## 4  0.052        999     0.175       NA 999
## 5  0.048         99     0.164       NA 999
```

Discuss with your neighbour!

What does this part do?

```
here::here(
    "data-raw",
    "D010",
    "dengue_data.csv"
  )
```

Why is it handy?

see “here! here!” - github link J. Bryan

Please do no do this in your code, anywhere:

```
data_dengue <- read_csv(
  "D:/r_projects/start_with_rmd/data-raw/dengue_data.csv",
  comment = "#")
```

Use the {here} package instead

What is typical about the missing values?

```
data_dengue[c(1:5), c(7:11)]
```

```
## # A tibble: 5 x 5
##   Mexico Philippines Singapore Thailand Venezuela
##    <dbl>       <dbl>     <dbl>    <dbl> <chr>
## 1 NA            999     0.059       NA missing
## 2 NA            999     0.059       NA missing
## 3  0.071        999     0.238       NA mising
## 4  0.052        999     0.175       NA 999
## 5  0.048         99     0.164       NA 999
```

Reference: Hadley Wickham - link

We could solve it like this

```
data_dengue$Venezuela[
  data_dengue$Venezuela ==  999] <- NA
data_dengue$Venezuela[
  data_dengue$Argentina ==  "missing"] <- NA
data_dengue$Philippines[
  data_dengue$Philippines == 999] <- NA
data_dengue$Philippines[
  data_dengue$Philippines == 990] <- NA
```

Discuss with your neighbour

Why is this syntax a bad idea?

```
data_dengue$Philippines[
  data_dengue$Philippines == 990] <- NA
```

DRY - Don't Repeat Yourself

1. When entering the 'same' code twice, it is time to start writing a function
2. Copying & pasting is error prone
3. Extend this to files, workflows and data-copies
4. Keep everything that belongs together in the same place:

"Start with Rmd"

```
RStudio Project --> RMarkdown file --> R-package
```

Functions in R

```
descriptive_function_name <- function(arguments, ...){
 if(some_condition_on_argument{
   message("message that condition meets argument(s)")
 } else {
   warning/stop("condition does NOT meet criteria ")
 }

 * some operations on the function arguments
 * some more calculations,
 * maybe reshaping the object or looping or transforming
 * ... = arguments that can be passed to functions
 used inside a function

 return(whatever_the_function_returns)
}
```

Putting the DRY principle to practice

The function below replaces certain values (`na_string`) in a vector for `NA`

```
replace_x_for_na <- function(x, na_string){
  x[x %in% na_string] <- NA # %in% is a special for match
  return(x)
}
```

Apply this function to one column of our data_dengue

```
data_dengue$Venezuela[1:4]
```

```
## [1] "missing" "missing" "mising"  "999"
```

```
replace_x_for_na(x = data_dengue$Venezuela,
                 na_string = c(99, 990:999, "missing", "mising
  as.numeric() %>%
  head(4)
```

```
## [1] NA NA NA NA
```

Apply our function to whole data_dengue dataframe

```
data_dengue_new <- data_dengue %>%
  purrr::map_df(
  replace_x_for_na,
  na_string = c(99, 990:999, "mising", "missing"))
data_dengue_new[c(1:5), c(7:11)]
```

```
## # A tibble: 5 x 5
##   Mexico Philippines Singapore Thailand Venezuela
##    <dbl>       <dbl>     <dbl>    <dbl> <chr>
## 1 NA             NA     0.059       NA <NA>
## 2 NA             NA     0.059       NA <NA>
## 3  0.071         NA     0.238       NA <NA>
## 4  0.052         NA     0.175       NA <NA>
## 5  0.048         NA     0.164       NA <NA>
```

Function Documentation

Let's assume we want to write a bit of documentation on how to use our `replace_x_for_na()` function

Documentation for functions

How it is done in R-packages Let's look at the documentation for the function `mean`

```
?mean
```

Writing you own documentation: Roxygen Comments

To construct this formal documentation structure we can use Roxygen Comments (#')

```
#' @title Change values into formal NA
#'
#' @param x A vector containing non-formal NA values
#' that need to be replaced by formal NA
#' @param na_string A vector of values indicating
#' which values need to be replaced by NA
#' @return A mutated vector, with NA as replacements
#' @example <a reprex goes here>
#' @export

<function definition>
```

Steps to build an R-package (from Rmd)

1. Create a package project `usethis::create_package(<pkg_name>)`
2. Extract functions from your Rmd file into the ~/R folder
3. Write/Check documentation
4. Add other package infrastructure
5. Add dependencies
6. Build documentation and vignettes
7. Write tests –> Test package
8. Check package
9. Build package (source/binary)
10. Publish package on Github and/or CRAN/Bioconductor

Common workflow

Step 1: create an R-package RStudio project

Many steps of building an R-package are supported by the `{usethis}` package

1. This will create a backbone for an R-package called 'bumblebee' in the root and will start up the RStudio project in a separate session
2. Here we will put this package inside an existing RStudio project, usually this is a bad idea (but now we can more easily monitor whats going on)

```
usethis::create_package(path = "bumblebee")
```

Populating the /bumblebee/R/ folder

Execute the following steps inside the ~/bumblebee project at the Console

1. Create a backbone function .R script for the function `replace_x_for_na()` with `usethis::use_r("replace_x_for_na")`
2. Copy the function definitions + added Roxygen documentation to this .R file

Generate documentation for the functions and the package

Run the following commands inside ~/bumblebee at the Console

```
## add package documentation
usethis::use_package_doc()
## build documentation for functions
devtools::document()
```

Add dependencies

```
## add the pipe es dependency so you can use it in you functi
usethis::use_pipe()
## add ggplot2 as a dependency
usethis::use_package("ggplot2")
usethis::use_package("dplyr")
usethis::use_package("readr")
```

Add a licence

Here we will use CC-BY as an example

```
usethis::use_ccby_license("Your Name")
```

Add raw-data

If you want to add a dataset, the raw-data can be put inside the ~/bumblebee/data-raw folder. The script that generates the dataset also goes inside the data-raw folder.

```
usethis::use_data_raw()
```

A short cleaning chunk

We can cleanup the data_dengue and put this code in the 'data-raw' folder

```
## put the raw dataset inside 'data-raw'
## copy the loading of the dataset code inside DATASET.R
## copy this code to the DATASET.R file

data_dengue_tidy <- data_dengue %>%
  purrr::map_df(
  replace_x_for_na,
  na_string = c(99, 990:999, "mising", "missing")) %>%
  gather(Argentina:Venezuela, key = "country", value = "cases"
```

Add /R/data_dengue_tidy.R to include this dataset in the package NAMESPACE

```
usethis::use_r("data_dengue_tidy")
```

Building the documentation

```
devtools::document()
```

Building the package

Click ‘Build’ –> ‘Install & Restart’

Add a vignette

To add full documentation:

```
usethis::use_vignette("bumblebee-demo")
```

You can then copy chunks from the current Rmd to this vignette. Of course you can leave out the function definitions because these will be avaialable from the package now.

Rendering the vignette

```
devtools::build_vignettes(pkg = "../bumblebee")
## load all to view the vignette
devtools::load_all(".")
## view the vignette
browseVignettes(package = "bumblebee")
```

Writing tests with `testthat()`

Tests are important because:

1. Things break
2. Versions change (R and Dependencies)
3. Inputs change

How to start with tests

Again: {usethis} makes it easy to start

```
usethis::use_test("replace_x_for_na")
## test for datset
usethis::use_test("data_dengue_tidy")
```

Test coverage

To see what still needs to be done ;-)

```
usethis::use_coverage()
```

Building and Checking package

Click:

- 'Build' –>
- 'More' –>
- 'Clean & Rebuild' –>
- 'Test Package' –>
- 'Check Package' –>
- 'Build Source Package' (Linux / McOS / Windows)
- 'Build Binary Package' (Windows only)

Integration with Git/Github

Initialize an empty repo in Github –>

In Terminal execute:

```
echo "# bumblebee" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:<gihub_username>/bumblebe
git push -u origin master
```

Commit changes locally –> push changes to remote