Tutorial 1: R and RMarkdown

Benjamin

10th April 2024

R Markdown documents are comprised of 3 primary elements:

1. the YAML or metadata at the top, separated by the 3 dashes `---`
2. code chunks, delineated by 3 back ticks and curly brackets ` ```{} `
3. plain text formatted in Markdown

Unlike a Jupyter notebook which is comprised of different types of cells, RMarkdown files only have a cell equivalent for code in the form of code chunks.

Task 0

You may have noticed a popup when you opened this RMD file that at the top stating "Package gapminder required but is not installed". This is because the gapminder package is not already in Noteable. You only need to install a package once, but every time you want to use you need to call it from the library. You can think of a package like a book - you only need to buy a book once from the store, but every time you want to access some information within it, you need to pull it off your bookshelf and open it. Buying the book from the store is equivalent to `install.packages("packagename")` in code while pulling the book off the bookshelf to use it is equivalent to`library(packagename)` in code.

You can either click the install button which will run `install.packages("gapminder")` for you, or run this code yourself in the console.

Task 1

The YAML at the top of the document is the **metadata**. Put your name in the YAML as well as today's date and knit your document to see what this has changed.

Rather than hard coding the date, you can use

> "2024-04-10 14:13:52"

Bonus To change the format of this date/time, wrap `Sys.time()` inside `format()` , e.g., `format('%H:%M %d-%b-%Y')`. Because you're already using double quotes in the date, you'll have to use single quotes inside the `format()` function. So it ends up like this:

> "Apr-2024"

Note the difference between back ticks (') and single quotation marks (')

Date-time formatting options:

Notation	Meaning	Example
%d	day as number	01-31
%A	weekday	Monday-Sunday

Notation	Meaning	Example
%a	abbreviated weekday	Mon-Sun
%m	month	January-December
%b	abbreviated month	Jan-Dec
%Y	4-digit year	2021
%y	2-digit year	21
%H	hour	12
%M	minute	56

Choose some of the date time formatting options and re-knit your document to see how they work.

After the global code chunk at the top under your YAML, it is good practice to have a code chunk where you call the packages you will be using in your document with the `library()` function.

```
#Sys.setenv(TZ="Europe/London") # set timezone (TZ argument) to whatever timezone you may be in, I am i
# for a list of TZ identifiers see here: https://en.wikipedia.org/wiki/List_of_tz_database_time_zones

library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.5
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.5.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
#install.packages("gapminder") # install the gapminder package
library(gapminder) # for our data
```

Task 2

Run the code in the libraries chunk either using the Run button, a key board shortcut (`Ctrl + Alt + R` on a Windows or `Ctrl + A + Enter` on a Linux and Windows to run the whole chunk), or using the run buttons in the code chunk itself.

Notice that when you load the tidyverse suite of packages there is a message output explaining the packages that were attached (and their version) as well as the conflicts. Because R packages are developed by various users, there are some functions with the same name across different packages. The conflicts section of the message explains which functions are being masked. So for example in line 2 `dplyr::filter()  masks stats::filter()` is telling you that when you use the `filter()` function without specifying the package of the function, it will automatically using the `dplyr` version of the function with the name `filter`, rather than the function from the `stats` package. This is because when two packages have a function of the same name, the package loaded *last* will find the function from earlier packages.

You can also work around this by explicitly telling R that you want to use the function from a specific package. This is where the double-colon `::` or namespace syntax comes in. Note that this is used in the message when loading the `tidyverse` package.

This is generally not an issue, but something to keep in mind if your code is not behaving as you expect - you could be accidentally using a different function than you intended to! So when these messages appear when loading packages, do not ignore them.

If you have any questions about this, ask one of the teaching team during the tutorial (or post on the discussion boards).

Task 3

Again in the libraries code chunk, click on the cog wheel next to the run buttons to hide messages when the document is knitted. Notice what code this adds to the code chunk. Knit the document again to see what changes this makes.

R packages often have sample datasets in them, which is quite a unique R concept. There are now some packages in Python and other languages that have followed suit. One of these is sample data sets is `gapminder`.

```
#save the gapminder data into the working environment
gap_data <- gapminder

# correlation for inline code example below
LePcor <- cor.test(gap_data$lifeExp, gap_data$pop)
LePcor
```

```
##
##  Pearson's product-moment correlation
##
## data:  gap_data$lifeExp and gap_data$pop
## t = 2.6854, df = 1702, p-value = 0.007314
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.01752303 0.11209600
## sample estimates:
##        cor
## 0.06495537
```

Task 4

Look in the global environment pane and double check that we now have an object called `gap_data` and `LePcor` listed there. If you want to see the data in a spreadsheet like manner, you can double click on the object `gap_data` or use the function `View(gap_data)`. Try it now!

```
#View(gap_data)
```

Task 5

Using the help documentation, find out what data the `gapminder` dataset covers.

```
help(gapminder)
??gapminder     # metadata
```

Hint you can use the help tab or the `?` notation

Optionally, you can also look up the documentation for the `View` function.

Task 6

In the RStudio environment there is a handy Markdown Quick Reference built-in. Pull this up by going to `Help` in the toolbar and selecting the quick reference guide.

Images

You can easily insert images into your R Markdown file either from a webpage or from a file in the same directory:

1. From a webpage

Figure 1: Beach Chairs

To get the link to a picture from a webpage go to your internet browser of choice, Images, and right click on the image and select "Copy Image Link".

Note: When knitting to pdf, you need to add another line of code in the YAML for LaTeX `graphicx` to pull an image from the internet and then save them locally in the same directory as the RMD file.

```
`output:
   pdf_document:
     pandoc_args: ["--extract-media", "."]`
```

2. From a file

`![alt text](file path.file extension)`

Note the back ticks are not needed in line 135, I have only included them so that there is not a knitting error, when completing the task remove the backticks as in line 124.

Hint: do not forget to add the R logo image to your Noteable file directory/the working directory of your Project if you are using Projects. I mentioned in the Python notebook that it is good practice to have a folder for your figures to keep things organised, but this will impact your file path. For example, if your file path was "figures/Rlogo.png" in plain language this means go to the "figures" folder and find the file called "Rlogo.png". However, this only works if the figures folder is in your working directory. If the figures folder were one folder up from your project working directory you would need to use "../figures/Rlogo.png". The .. is how you navigate back or up and out of a folder.

Task 7

Insert the image of the R logo below. Afterwards, try to insert an image from the a URL as well!

Hint Remember that R cannot find an image on your computer as we are using Noteable, which is a cloud-based service. You will need to put the image on your Noteable file directory and tell R where to find it.

Inline Code (Optional)

You can also use R code directly in text. For example:

> There are 1704 observations in the data set.

This can be an incredible time saver if you are creating a report where the data itself might still be coming in. Rather than needing to manually update descriptive information such as how many observations there are in the data set, let R do it for you!

I also will often use this when reporting in text results of analyses. For example:

> The correlation between life expectancy and population is $r(1702) = 0.065$, $p = 0.007$

Mathematical notation

You will notice above that I used something else in combination with the inline r code - mathematical notation! This is based on LaTeX. In side a text chunk, you can use mathematical notation if you surround it by dollar signs `$` for "inline mathematics" and `$$` for "displayed equations". This post by R Pruim is my go-to resource for mathematical notation in R Markdown.

Top tip - document and chunk outlines

Once your R Markdown document starts to get quite long (you've been working hard!), with several different headings and perhaps various subheadings too, it can start to get slightly tricky to quickly scroll and find the various sections in your raw document.

To help with this, RStudio have a helpful tool, the **Show document outline** (which is the button with various lines stacked on top of each other) option which you can find near the **Run** menu. This will show you an interactive outline of your document by the different headers/sections you have created.

This lets you quickly jump between sections in your document and see the outline of your structure at a glance. A similar interactive outline feature can be found at the bottom of your document/source pane, and code chunks are included here too.

Task 8

Find the interactive document outline and outline including code chunks. Click on a few of the different sections to how you can navigate using these outlines.

Output types and their functionalities

HTML

In general, when knitting to an `html_document` there are more interesting functionalities. Indeed, Markdown was originally designed for HTML output so the HTML format has the richest features.

- floating table of contents
- code download output option
- code folding (hide or show)
- tabbed sections

You can have a look at some of the other options when knitting to HTML in the HTML document chapter in R Markdown: The Definitive Guide

Chunk names are not necessarily required to knit, but are good practice and help to support more advanced knitted approaches. Chunk labels should be

- unique
- do not use spaces, rather use dashes (-)
- use alphanumeric characters (a-z, A-Z and 0-9), because they are not special characters and will surely work for all output formats
- spaces and underscores in particular, may cause trouble in certain packages, such as `bookdown`.

We do not cover the `bookdown` package as part of this course, but it extends the functionality of RMarkdown to allow for figures and tables to be easily cross-referenced within your text (among other things).

Task 9

Explore some of the cool customizations for HTML documents in the R Markdown Definitive Guide book link. Choose one or two of them and add them to this document and knit to see how they work!

Some suggestions are a floating table of contents or the code folding.

PDF

PDFs in R Markdown also have a variety of features, but not nearly as many as HTML documents. To find out more, check out the PDF document chapter in R Markdown: The Definitive Guide

Task 10

Knit the document now also to a PDF as well as HTML, perhaps including customizations for each document type.

Hint 1 if you do not want to type in the YAML yourself, you can click the arrow under the knit button and select PDF and the PDR output option will automatically be added to the YAML for you.

Hint 2 you will likely run into an error when knitting to PDF. Try to read the error and see what you can understand may be causing the issue. Look on line 126 of the notebook for a hint as to why. Any questions ask one of the tutors during the tutorial or post on the discussion boards!

Word

For completeness I have included a section on Word documents, but they will not be used in this course as a file output type.

The most notable feature of Word documents in R Markdown is the ability to create a "style reference document". To find out more, including a short video on how to create and use a reference document, check out the Word document chapter in R Markdown: The Defintive Guide

Themes (Optional)

In general, there are a variety of theme options available to use with R Markdown HTML documents.

The following packages are not installed already on Noteable. So in order to use `prettydoc` or `rmdformats`, you will need to first install the packages and then call them via the `library()` function.

`prettydoc`

The `prettydoc` package includes a variety of other theme options when knitting to HTML: https://prettydoc.statr.me/index.html

Once you have installed the package, you can open a prettydoc formatted document from "From Template" tab when choosing to create a new R Markdown file.

To use a `prettydoc` theme not from a template, you need to edit the YAML accordingly:

```
`output:
    prettydoc::html_pretty:
    theme: cayman`
```

Note: When using `html_pretty` engine for themes, `code_folding`, `code_download`, and `toc_float` are not applicable.

`rmdformats`

The `rmdformats` package includes a variety of other theme options when knitting to HTML: https://github.com/juba/rmdformats. Some themes allow for things like a dynamic table of contents, but not all of them. See the "Features matrix" table on the above webpage for more info.

Similar to above, once you have installed the package, you can open a rmdformts formatted document from "From Template" tab when choosing to create a new R Markdown file.

To use a `rmdformats` theme not from template, you need to edit the YAML accordingly:

```
`output:
    rmdformats::robobook`
```

Reproducibility!

In practice, depending on your audience, you will need to decide to show your code or not. It is unlikely that you will want to show the code used to produce your analysis, tables, or figures to an audience unfamiliar with R and would therefore set `echo = FALSE` in the set-up chunk. For this course, though, and in particular for the programming assignment, you will need to set `echo=TRUE` so that we can see your code and the product of that code.

While it can take up space, it is good practice to finish a document calling the `sessionInfo` function, which lists all of the packages you used, their versions, and more.

```
## R version 4.1.3 (2022-03-10)
## Platform: x86_64-conda-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.4 LTS
##
## Matrix products: default
## BLAS/LAPACK: /opt/conda/lib/libopenblasp-r0.3.21.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
```

```
##
## other attached packages:
##  [1] gapminder_1.0.0 forcats_0.5.2   stringr_1.5.1   dplyr_1.0.10
##  [5] purrr_0.3.5     readr_2.1.3     tidyr_1.2.1     tibble_3.1.8
##  [9] ggplot2_3.3.6   tidyverse_1.3.2
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.2.0    xfun_0.34           haven_2.5.1
##  [4] gargle_1.5.2        colorspace_2.0-3    vctrs_0.6.5
##  [7] generics_0.1.3      htmltools_0.5.3     yaml_2.3.6
## [10] utf8_1.2.2          rlang_1.1.3         pillar_1.8.1
## [13] withr_2.5.0         glue_1.6.2          DBI_1.1.3
## [16] dbplyr_2.2.1        modelr_0.1.9        readxl_1.4.1
## [19] lifecycle_1.0.3     munsell_0.5.0       gtable_0.3.1
## [22] cellranger_1.1.0    rvest_1.0.3         evaluate_0.17
## [25] knitr_1.40          tzdb_0.3.0          fastmap_1.1.0
## [28] fansi_1.0.3         broom_1.0.1         scales_1.2.1
## [31] backports_1.4.1     googlesheets4_1.0.1 jsonlite_1.8.3
## [34] fs_1.5.2            hms_1.1.2           digest_0.6.30
## [37] stringi_1.7.8       grid_4.1.3          cli_3.4.1
## [40] tools_4.1.3         magrittr_2.0.3      crayon_1.5.2
## [43] pkgconfig_2.0.3     ellipsis_0.3.2      xml2_1.3.3
## [46] reprex_2.0.2        googledrive_2.0.0   lubridate_1.9.3
## [49] timechange_0.3.0    assertthat_0.2.1    rmarkdown_2.17
## [52] httr_1.4.7          rstudioapi_0.14     R6_2.5.1
## [55] compiler_4.1.3
```

This documentation of your environment configuration is not only good practice, but also proves invaluable in any future debugging (fixing errors or bugs in your code), ensuring compatibility with ever evolving software versions.

Well done! You have completed all of the tasks for the RMarkdown notebook. If you have not done so yet, now move to the Python notebook.

Dr Brittany Blankinship (2024)