Producing dynamic reports

Demonstration

2022-06-14 11:24:21

Contents

Introduction

This is our dynamic report. The report will teach you how to make the report.

- work through in order
- ask questions if you get stuck
- ask about how you might adapt this to suit your needs

Getting set-up

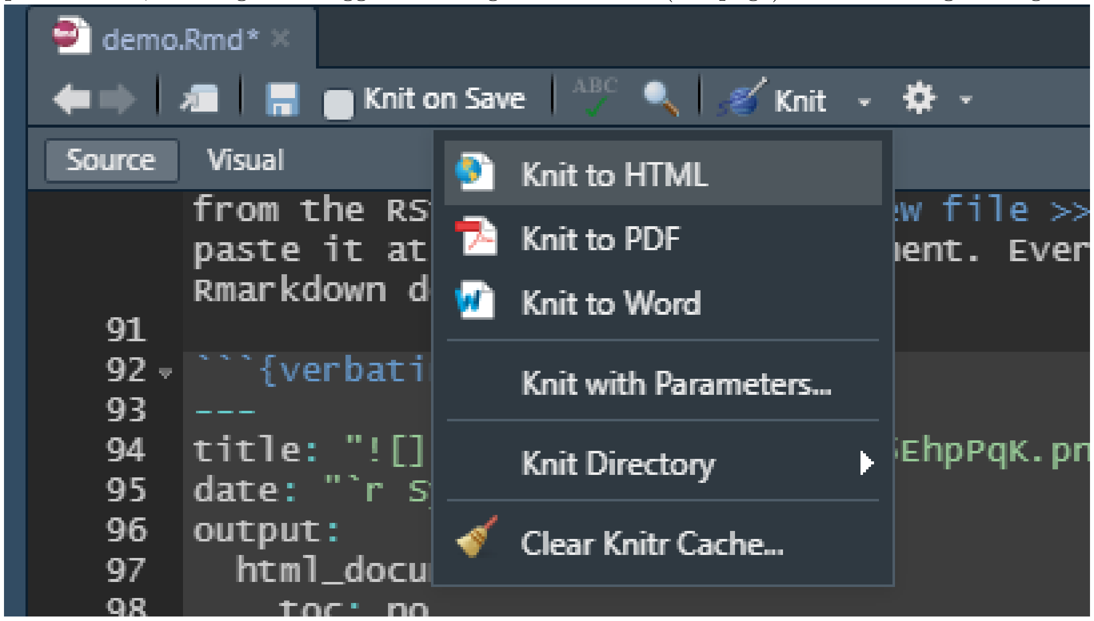
Starting an Rmarkdown document

There's a bit of preamble at the head of Rmarkdown documents that controls how they are built. For now, I'll suggest that you skip trying to figure it all out, and just use the header from this document. I'll give more details about setting up your Rmarkdown document at the end of this demo.

Knitting

We definitely don't want our end users to see our RMarkdown code. So the .Rmd file that you've been working on needs to be converted into a suitable output format. We call that conversion *knitting* (after the R package *knitr* that manages the conversion). To knit your file, you can either:

- select the Knit option from the menu at the top of the code pane. You can select any of the output formats, although I'd suggest sticking with HTML (webpage) while working through the demo.

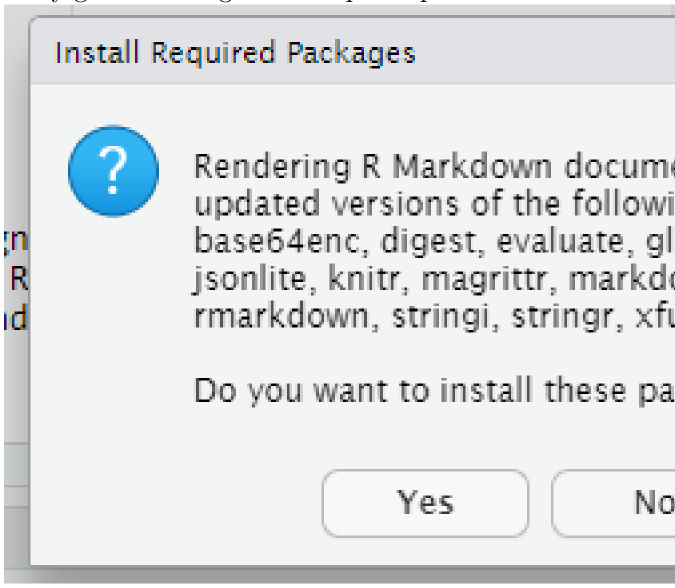


~~~
from the RS                          w file >>
paste it at                          ent. Ever
Rmarkdown d

```{verbati
---
title: "![]                          EhpPqK.pn
date: "`r S
output:
  html_docu
    toc: no
~~~

- alternatively, press ctrl+shift+k to knit to the last used format

The first time that you knit your document in RStudio Cloud, you may get a warning about required packages. Just agree - you'll need them to knit your document properly.


Install Required Packages

Rendering R Markdown docume
updated versions of the followi
base64enc, digest, evaluate, gl
jsonlite, knitr, magrittr, markdo
rmarkdown, stringi, stringr, xfu

Do you want to install these pa

Getting R ready

Once you've got the header in place, we need to do a tiny bit of setting R up properly.

```
library(tidyverse)
library(lubridate)
library(NHSRplotthedots)
```

We load three packages (link to eventual NHS-R statement on tools and packages) here to help us do some useful work with our data. These are all *very* useful for health and care projects, so we'll briefly introduce them here:

- tidyverse - large ecosystem of tools designed for data science
- lubridate - helps with handling dates and times
- NHSRplotthedots - NHS-R community tool for making XmR charts

You may need to install some or all of these packages. If you need to do this, you can add the line `install.packages("PACKAGENAME")` to your file, run that line (ctrl + enter), and then remove the line of code. Or install via `Tools > Install packages...`.

Now that's working, we can skip ahead to the interesting part where we get start to work on our report..

Rmarkdown

Rmarkdown is a simple markup language. You might have used it before without really knowing what it was called. If you've ever posted on a Wordpress site or web forum with basic text formatting options available, the chances are that you've already done at least a tiny bit of markdown. Just to give you an example, surrounding a word with asterisks - like `*italics*` formats it in *italics*. Pairs of asterisks either side of the word formats that word - like `**bold` - formats the text as **bold**. And so on.

This kind of very basic formatting is known as Markdown. While it's simple, it's usually sufficient to allow the production of neat and clear documents that are easy to ready. Rmarkdown is just Markdown with a few extras to allow you to embed R code directly into the text that you're writing. That R code allows you to do more-or-less any data analysis that you might like and include the results in your markdown-formatted document. If this is all new to you, you might like to have a quick look at some of the excellent introductory R/Rmarkdown resources:

italics: `*italics*`

bold: `**bold**`

links: `[links](https://google.com)`

Heading levels

```
# Level 1
## Level 2
### Level 3
...

![Linking images](https://i.imgur.com/5EhpPqK.png)

- dashes at the start of lines make bullet-points
- like so
```

- dashes at the start of lines make bullet-points
- like so

```
1. Numbers at the start of lines make numbered lists
2. like so
```

1. Numbers at the start of lines make numbered lists
2. Like so

Do have a look at the Rmarkdown in the code pane, and feel free to play around with it. Once you've had a look at some of the Rmarkdown in practice, we're ready to move on to get some data.

Get data

This ~~feels~~ is easy to do in practice. Instead of opening a data file directly (as we might when using Excel), and then inspecting and analysing it, we instead load all the data in that file into R.

We load the data using the following command:

```
demo <- read_csv("demo.csv")
```

- `read_csv` is the function that opens the .csv data file
- `demo.csv` is the path to our data file
- `<-` assigns the data that read_csv pulls out of the file to…
- `demo` - the tibble that contains our data

Explain chunks

Because there are lots of different types of data out there, there are lots of ways of loading data into R. `read_csv` is the most commonly-used one, because .csv files - **c**omma **s**eparated **v**alues - are widely used to share data. The other main function used to load data is tidyverse's `read_excel`, which works similarly to `read_csv`.

Rather than explain how this works here, I'll encourage you to read the relevant manual pages, which you can do with the command `?read_excel`. You can run this using ctrl+enter in RStudio.

At the end of this process, we can have a look at the first few rows of our data using the `head()` command:

```
head(demo)
```

```
## # A tibble: 6 x 4
##   date       count1 count2 count3
##   <chr>       <dbl>  <dbl>  <dbl>
## 1 01/05/2022     11     93     82
## 2 02/05/2022     15    114     99
## 3 03/05/2022     18     28     10
## 4 04/05/2022     19    107     88
## 5 05/05/2022     21    140    119
## 6 06/05/2022    170    217     47
```

This command shows us the first few rows of our data. From the .csv file, we have converted the data into a *tibble*, which is the name for the data structure used by the package `tidyverse`. In this tibble (called `demo`), each row contains four columns. One holds date, and the remaining three hold of counts of some imagined values from those dates. We can ask R to tell us other properties of the `demo` tibble too. For example, we can count the total number of rows with:

```r
nrow(demo)
```

```
## [1] 68
```

Or we can pull out the column headings with:

```r
colnames(demo)
```

```
## [1] "date"   "count1" "count2" "count3"
```

Note that in the couple of examples above, we've just asked R to show us the number of rows/column names. We can also save these findings, which we might do if we want to use them later for another purpose. Just for instance, we can make a new variable *demo_length* containing the number of rows in data with:

```r
demo_length <- nrow(demo)
```

And we can refer back to the value of *demo_length* whenever we like:

```r
demo_length
```

```
## [1] 68
```

Or do things to that value - like multiplying it by five:

```r
demo_length * 5
```

```
## [1] 340
```

We can also create new variables from more complicated operations on our original tibble. For instance, we can extract a whole column using `tibble$column`, and then add all the entries in that variable together using `sum()`:

```r
sum_count3 <- sum(demo$count3)
sum_count3
```

```
## [1] 4792
```

I think that's enough demonstrating for now. There's one an important problem with our tibble: our dates aren't proper dates yet. `read_csv` doesn't know that the values in the date column actually are dates. We can tell that by looking at the `<<chr>>`, which stands for character in when we looked at `head(demo)` above. Character is the way that R refers to text values. But we definitely want those values to be stored properly as dates, and doing that conversion will be our next exercise.

Clean up data

Time for another R chunk:

```
demo <- demo %>% mutate(date = dmy(date))
```

There's a bit to unpack here. It makes most sense (to my mind, at least) to read this code backwards. From right to left, then:

- `dmy(date)` is a lubridate command which takes whatever's in the bracket, and tries to convert it into a proper date. It works by assuming that it'll be a day value (d), followed by a month (m), followed by a year (y). There are also related commands like `ymd()` to be used when the dates given in our data are formatted differently.
- `mutate(date =...)` takes the result of the `dmy()` above, and uses it in a tidyverse command `mutate`. This is a really important and powerful tool, which makes new columns in tibbles from existing columns. Here, we tell `mutate` to make a new column called `date` by taking the result of the `dmy(date)` above.
- `%>%` the pipe! This is a way of chaining commands together. In this case, writing `demo %>% mutate...` tells R to take the demo tibble, and do the mutate command to it. The advantage here is that we can string `%>%`s together, to do many operations in sequence on some set of data.
- finally, we assign the modified demo tibble back on top of itself using `<-`. Essentially, `demo <- demo ...` tells R to overwrite the starting version of demo with the updated version containing proper dates.

That simple exercise introduces several new concepts. **Question:** pause here? Explain again? Another exercise?

Now that we have some data loaded and cleaned up, we're ready to use it to start writing our report.

Getting text to update to reflect data

As a refresher, let's have a look at the first few rows of the demo tibble using `head()`, and also print out the total number of rows using `nrow(demo)`:

```
head(demo)
```

```
## # A tibble: 6 x 4
##   date       count1 count2 count3
##   <date>      <dbl>  <dbl>  <dbl>
## 1 2022-05-01     11     93     82
## 2 2022-05-02     15    114     99
## 3 2022-05-03     18     28     10
## 4 2022-05-04     19    107     88
## 5 2022-05-05     21    140    119
## 6 2022-05-06    170    217     47
```

```
nrow(demo)
```

```
## [1] 68
```

This data will change as the data in the demo.csv file changes. You can show yourself how this works as follows:

1. Knit this document now (menu at top middle of code pane or ctrl+shift+k)
2. Have a look at this section again - you should see that our original data has 68 rows
3. Now edit the demo.csv file. To do this, click on the demo.csv file in the `Files` pane, and select `View File`. This should open the data file in the code pane.

4. Now delete a couple of rows of your choice, before saving and closing the file.
5. Now re-knit the document again. The data now has 68 rows, rather than the original 68 rows.

Is useful, but looks nasty

We can also produce summary text inline. For example, we can count up all of the rows here, and say that we have a total of 68 records. Or we can add all the entries in the count1 column (a total of 6643). Or find out that the maximum value in count3 is 141. Or we can say that on the first day of this range (19113), the value of count2 was 93. Or that the mean (average) of count3 was 70.5.

I'll say it again: **change the data, and all this stuff will update automatically**. No need to copy and paste at many locations. For example, you might like to include a date in your report describing which month the data is about. You can do this easily: the latest date in this data comes from July. Change the data, and this report will update - go on, try it!

Doing data processing

We can do useful stuff like add the columns together:

date	count1	count2	count3	total
2022-05-01	11	93	82	186
2022-05-02	15	114	99	228
2022-05-03	18	28	10	56
2022-05-04	19	107	88	214
2022-05-05	21	140	119	280
2022-05-06	170	217	47	434

(again, just showing the first few rows of data).

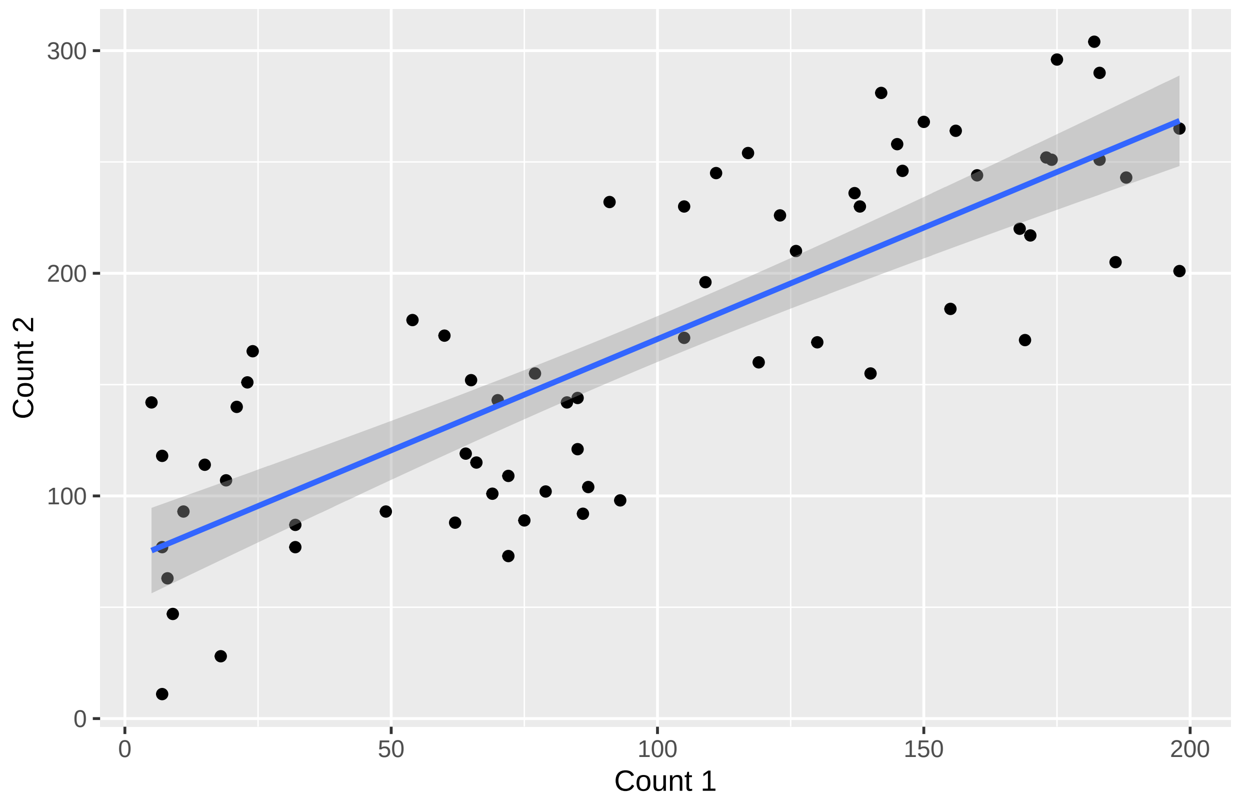
We can also summarise tables of data:

count1	count2	count3	total
98	168	70	336

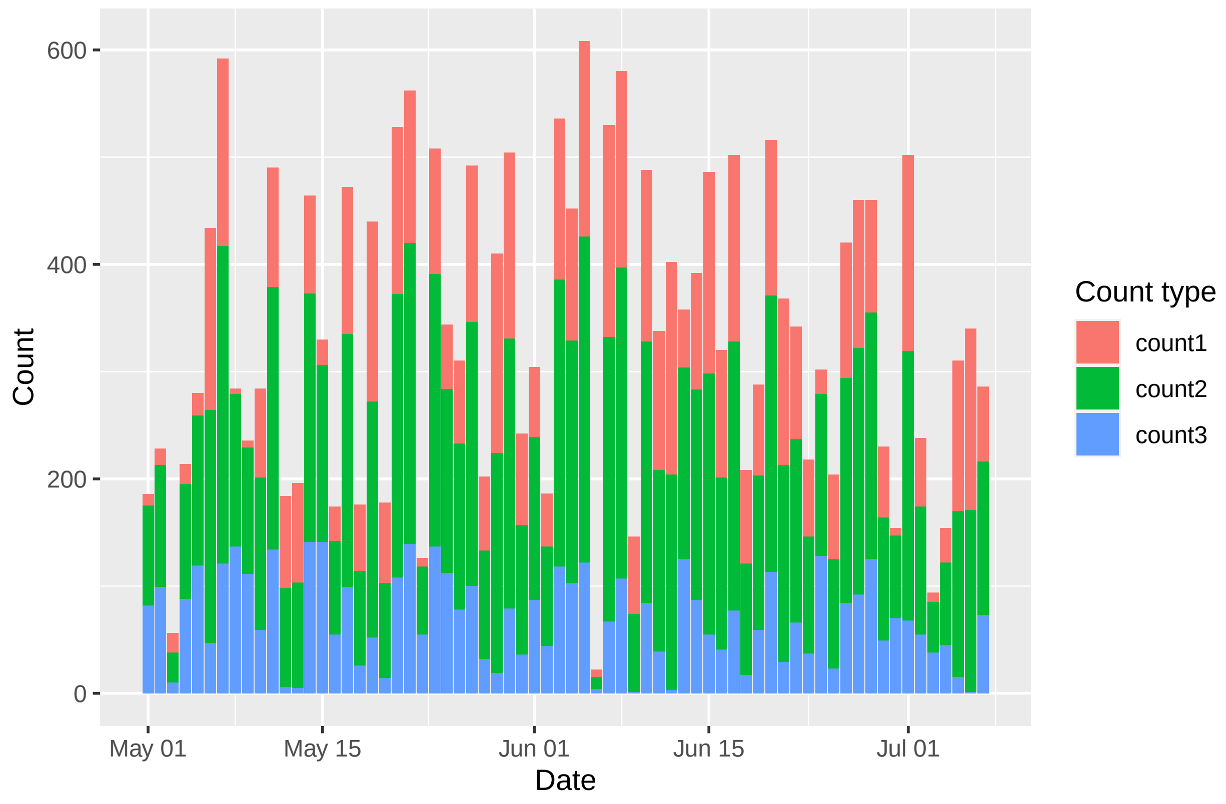
or re-name columns in our tables:

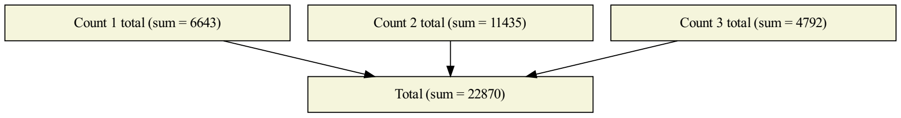
Mean of count 1	Mean of count 2	Mean of count 3	Mean of total
98	168	70	336

Scatter plot of count 1 against count 2 with linear regression



Stacked bar of counts by date




Count 1 total (sum = 6643)
Count 2 total (sum = 11435)
Count 3 total (sum = 4792)
Total (sum = 22870)

More details

Rmarkdown document settings

The easiest option is to start a new Rmarkdown document direct from the RStudio menu `File >> New file >> R Markdown...`. Alternatively, you can copy the header from this document, and paste it at the top of your document. Everything between the pair of three dashes: `---` is concerned with setting up this Rmarkdown document:

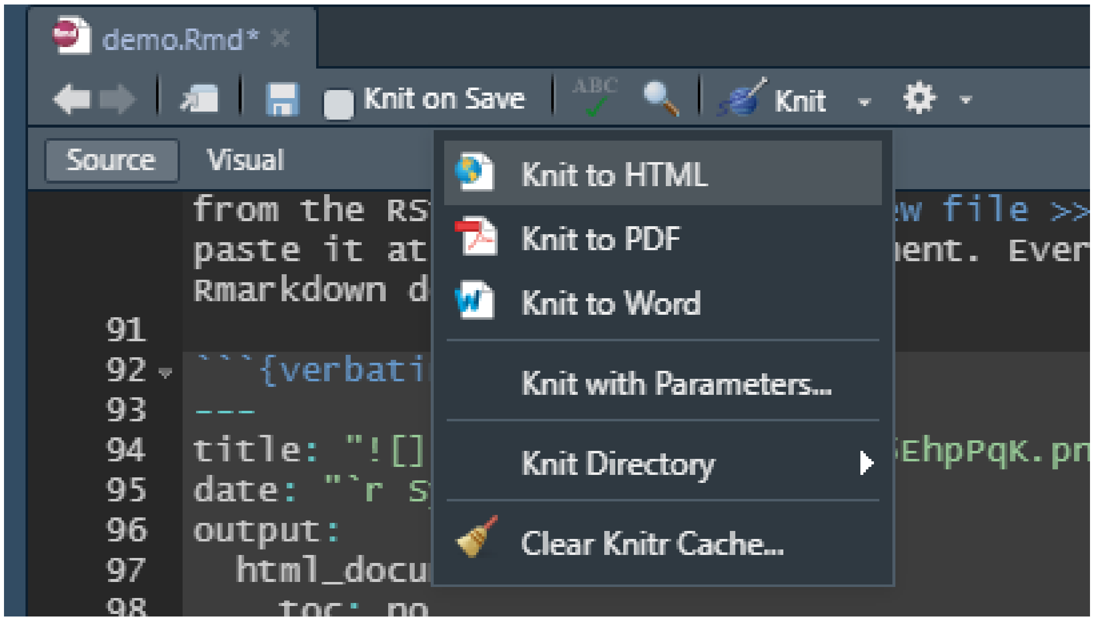
```
---
title: "![](https://i.imgur.com/5EhpPqK.png) Dynamic report demo"
date: "`r Sys.time()`"
output:
  html_document:
    toc: no
    toc_depth: 2
    number_sections: no
    toc_float:
      collapsed: no
  pdf_document:
    toc: yes
    toc_depth: '2'
    pandoc_args: ["--extract-media", "."]
  word_document:
    toc: yes
    toc_depth: '2'
---
```

You might notice that there are a couple of differences between the header from this document and the default version that RStudio gives you. I've added a link to the KIND Learning Network banner image as part of the title. If you're thinking about adapting this report to suit your purposes, you can easily change both of those by editing the URL (and the title text) in the following line:

```
title: "![](https://i.imgur.com/5EhpPqK.png) Dynamic report demo"
```

More info on YAML editing in chapter 2 of the excellent R Markdown Cookbook.

Most of this header block is made up of options for the different output formats (html, word, and pdf). In RStudio, you can select which output format you'd like your report to be rendered as:



~~~
from the RS                       w file >>
paste it at                       ent. Ever
Rmarkdown d

```{verbati
---
title: "![]                       EhpPqK.pn
date: "`r S
output:
  html_docu
    toc: no
~~~

Try these out now - this demo report should knit to each of the three formats without any problems. I'd recommend sticking with html while you're working on a report, though, just because it tends to render most quickly, and plays nicely with other aspects of the workflow while you're writing.

One thing to watch out for when tweaking these options: indentation and white-space matters. Each of the three output formats (html_document, pdf_document, and word_document) are indented from the *output:* line, and each of the output format options are indented too. A common cause of documents not knitting (see below) are small errors in the indenting in the header.

You might notice another tweak that I've made to the pdf output options for this document, which is to add the line:

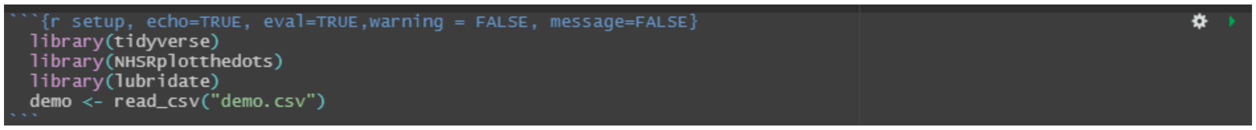
```
    pandoc_args: ["--extract-media", "."]
```

So what's next?

Alternatives In the R/Rmarkdown universe, Flexdashboard looks great: there's an example and a walk-through by the Miller Lab

More on R code chunks

Code chunks in Rmarkdown start and end with a triple backtick:



~~~
```{r setup, echo=TRUE, eval=TRUE,warning = FALSE, message=FALSE}
  library(tidyverse)
  library(NHSRplotthedots)
  library(lubridate)
  demo <- read_csv("demo.csv")
```
~~~

There are then a few options that you can set for the code chunk. Here:

- r: telling Rmarkdown to interpret this code as R. There are other languages that you can use in Rmarkdown, which is one of the strengths of building reports in this way.
- setup: the chunk label. This can be anything you like, but no duplicates are allowed. Very useful for navigation in more complicated reports
- echo=TRUE: whether to show this code in the report. If this is set to FALSE, you'll just see the output of the code (more on this below)
- eval=TRUE: whether or not to run the code. If eval=FALSE, the code won't do anything other than appear in the report (useful if you're explaining how something works - like this report!)
- warning = FALSE: whether to show warnings in the report if something goes wrong with your code
- message=FALSE: whether to show information messages about how your code is running

Lots more info in the Rmarkdown Cookbook

you can also use `knitr::opts_chunk$set`, which is a really powerful way of managing chunk options - but that's a bit beyond the scope of this demo.