University of Glasgow

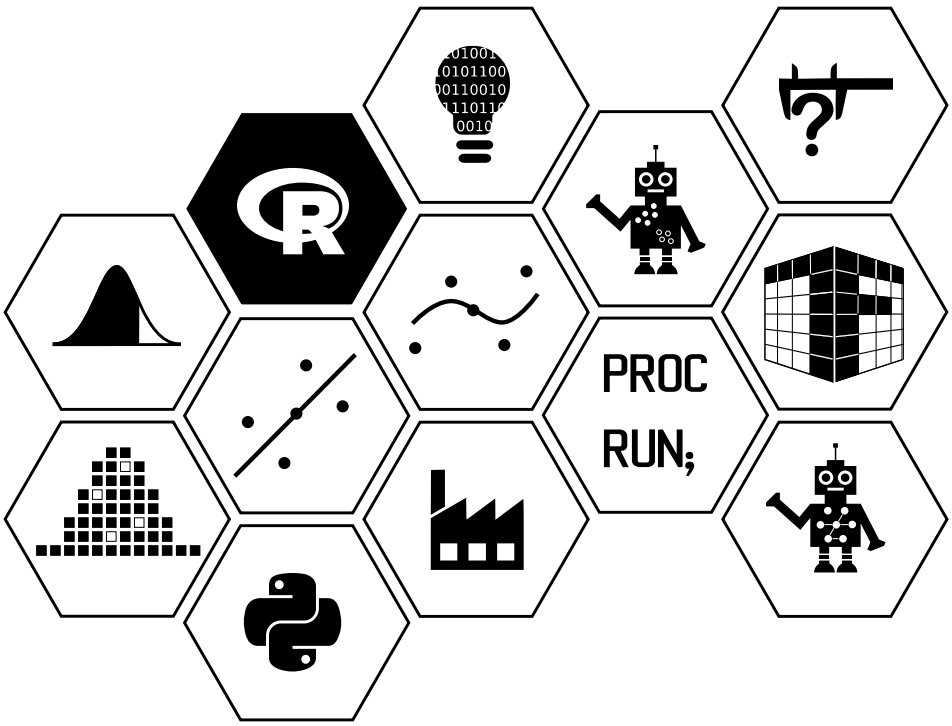
R Programming/ Statistical Computing

Craig Alexander

Academic Year 2023-24

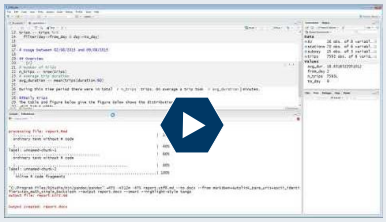
Week 11:

Automated reporting using knitr and R Markdown



DATA ANALYTICS GLASGOW

Automating reports



Using knitr and R Markdown

https://youtu.be/bvmFjpWD4S8

Duration: 9m22s

Why bother?

You might be used to running analyses in R (or some other software) and then copying and pasting the results (tables, plots, etc.) into a report written in Microsoft Word (or other software).

Though this might seem to be the simplest and quickest way of putting together a report, there are good reasons to move to a more unified (and more reproducible) way of producing reports.

- First of all, you might make mistakes when copying and pasting (or even retyping) the results into the Word document.
- Updating the report to take into account additions to or changes in the data can be a quite laborious process if you have replaced all the results you have copied into the document. The same applies if you find a mistake in your code and you need to rerun your analyses.
- If you want your report to be reproducible (by others or future you) you need to archive the R script that you have used to generate the report together with the report. I am sure it has already happened to most of you that you had update a graphic in a report and you just can't find the code you have used to generate it.

In this unit we will look at knitr, an R package for "weaving" R code into a "tangled" document, so that the results are computed and inserted automatically.

Literate programming

The idea of literate programming goes back to Donald Knuth:

> Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to humans what we want the computer to do.

Literate programming is based on the idea of combining code and prose into a single document. Initially this was done with the idea of better explaining the context and inner working of a programme, but the same idea helps with automatically generating reports. Rather than copying and pasting in a table or a plot, we include the R commands used to generate that table or plot in the report itself.

This has many advantages …

- We avoid the many pitfalls associated with manually inserting results into a document.
- When the underlying data changes (or we fix an error in our R code), all we need to do is recompile the report.
- It will be easy to parametrise a report, so we can generate several reports from one source.
- All we need to do is archive the one source file. This helps with making your work easier to reproduce.

Even if the main focus in data science is on automated report generation, we should not forget about the original idea of literate programming. If you write an R script performing a complex analysis, combining code and prose into a single document is an ideal tool for explaining your reasoning as to why you performed your analysis the way you have performed it. It also make it easy to insert an interpretation of the results (unfortunately, this interpretation won't update automatically when the underlying data changes). All this will help others (and future you) make better sense of your analysis. You don't want to come back to your code in a year's time and think "it made sense to me at the time, but now I have no clue what it does". This of course happens to all of us, but we would rather want to avoid it!

How will this work?

All we need to do is agree with the computer on a convention of what is to be treated as code and what is to be treated as prose. This is typically done by using tags to delineate code. What the tags are depends on the type of document we use.

Fortunately or unfortunately, including R commands straight into a Word document is difficult. Word won't call R in the way we want and R will struggle with Word documents being a quite complex document format.

To avoid such issues, it is easier to "weave" R code into text-based document formats. The most popular choices are …

- Markdown is a simple document format using plain text formatting. Many websites (like WordPress, Stack Overflow or GitHub) allow users to write content in Markdown. It is easy to convert Markdown documents to a variety of formats (including Word documents) using Pandoc.
- LaTeX is another (and much more complex and powerful) text-based format for authoring documents.
- HTML documents are also plain text instructions, so will be easy to insert code chunks.

We will first look at Markdown and then quickly look at inserting code chunks into LaTeX and HTML (for those of you familiar with those formats).

Using knitr and R Markdown

Markdown

Markdown is a simple document formatting language, which is almost WYSIWYG ("what you see is what you get"). Below is an example document written in Markdown.

````
# This is the first section

## This is a subsection

This text is an underlined heading
----------------------------------

This is a paragraph of text using *italics*, **bold** and `verbatim code` text.

This is a new paragraph with a [link to the University of Glasgow](https://www.gla.ac.uk).

* This is an item in an unordered list
* This is another item

Below is a table ...

|Column 1|Column2|
|--------|-------|
|123     |456    |
````

The simplicity of Markdown makes it very easy to automatically convert a Markdown document to a variety of other formats (HTML/Word/PDF documents, presentations, etc.).

Including chunks of R code

We can include an R code chunk in a Markdown document using the following syntax.

````
```{r #chunk options go here}
#R code goes here
```
````

R will run the code and depending on the chunk options (see below) show the R code and/or output. By default, both the R code and output will be shown.

Example 1 (A simple first R Markdown document).

The R Markdown document ...

````
Let's generate a random sample from the normal distribution.

We then look at some summary statistics and plot a histogram.
```{r}
x <- rnorm(100)
summary(x)
hist(x)
```
````

... generates the output shown below.

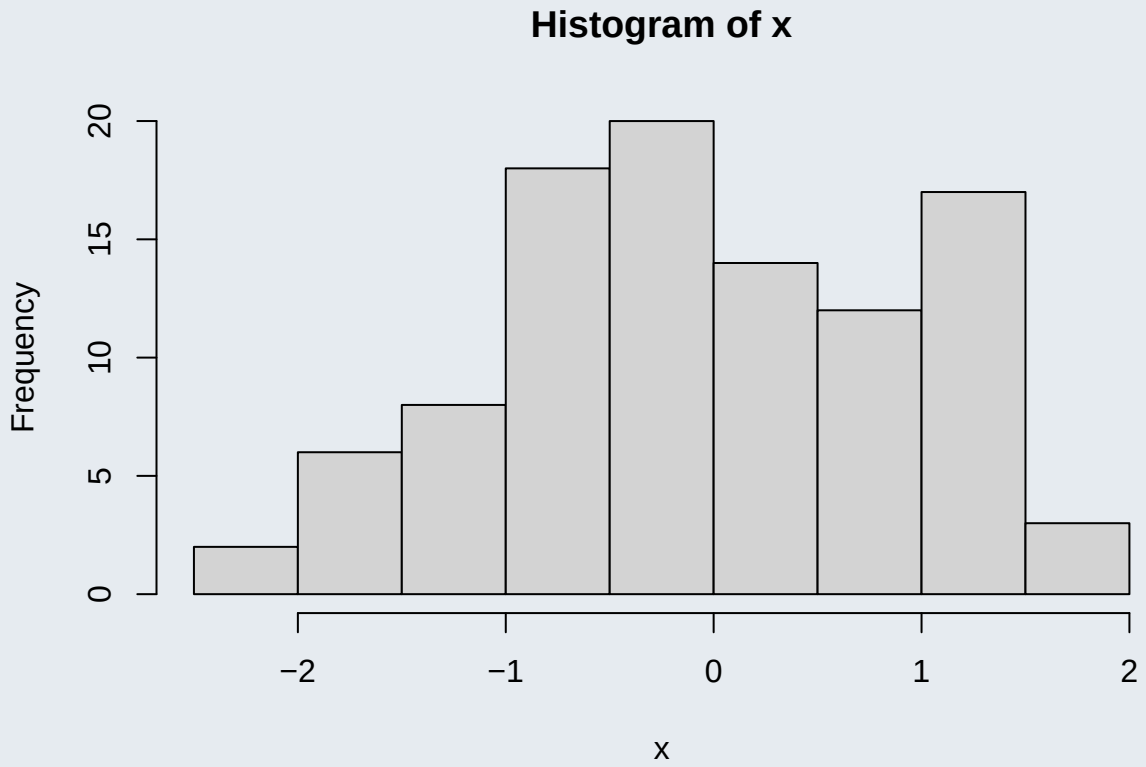
Let's generate a random sample from the normal distribution.

We then look at some summary statistics and plot a histogram.

```
x <- rnorm(100)
summary(x)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.19203 -0.78946 -0.04468 -0.03328  0.75008  1.73630
```

```
hist(x)
```



The R Markdown file for this example and the other examples in this unit is available in the zip file with all R Markdown files for this week.

Inline R expressions

Sometimes you want to include an R expression within a paragraph of prose. You can do this using inline R expressions. The syntax is

which can be placed inside paragraphs of text.

Example 2 (Inline R expressions).

The R Markdown document …

```
Let's draw a single random number x.

```r
x <- rnorm(1)
```
The random number x is -0.7677827.
```

… generates the output shown below.

Let's draw a single random number x.

```
x <- rnorm(1)
```

The random number x is 0.6828223.

Chunk options

Labelling chunks It is generally a good idea to label chunks. This makes it easier to locate errors. Chunks can be labelled by simply including the desired label name as the first argument.

```
```{r chunk-label}
#R code goes here
```
```

What to show? The table below shows the options which control whether code is being run and what is shown in the document.

Option	Show code	Run code	Output	Plots	Messages	Warnings
`eval=FALSE`	(Yes)	No	No	No	No	No
`include=FALSE`	No	(Yes)	No	No	No	No
`echo=FALSE`	No	(Yes)	(Yes)	(Yes)	(Yes)	(Yes)
`results="hide"`	(Yes)	(Yes)	No	(Yes)	(Yes)	(Yes)
`fig.show="hide"`	(Yes)	(Yes)	(Yes)	No	(Yes)	(Yes)
`message=FALSE`	(Yes)	(Yes)	(Yes)	(Yes)	No	(Yes)
`warning=FALSE`	(Yes)	(Yes)	(Yes)	(Yes)	(Yes)	No

The option `error` controls what `knitr` will do when it encounters an error: if it is `FALSE` then `knitr` will stop if it encounters an error (default behaviour in `rmarkdown`). If `error` is `TRUE`, then `knitr` will continue if it encounters an error and show the error message in the document. This is rarely useful, unless you want to illustrate the error message.

Other useful options You can add a caption to plots generated using the option `fig.cap="Figure caption"`. The options `fig.width` and `fig.height` let you control the size of the plot.

Global options You can use `knitr::opts_chunk\$set(option=value)` inside a code chunk to change default options for subsequent chunks.

Including data tables in the report

If you want to include tables showing data from R, you can use the function `kable` from `knitr` to produce these tables.

Example 3 (Data tables).

The R Markdown document …

````
We print the first four rows of the iris data set.

```{r echo=FALSE}
library(knitr)
kable(iris[1:4,])
```
````

… generates the table shown below.

We print the first four rows of the iris data set.

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa

You can pass on additional arguments to `kable` to change the appearance of the table.

There are a variety of R packages providing more sophisticated functions for creating tables such as stargazer or xtable.

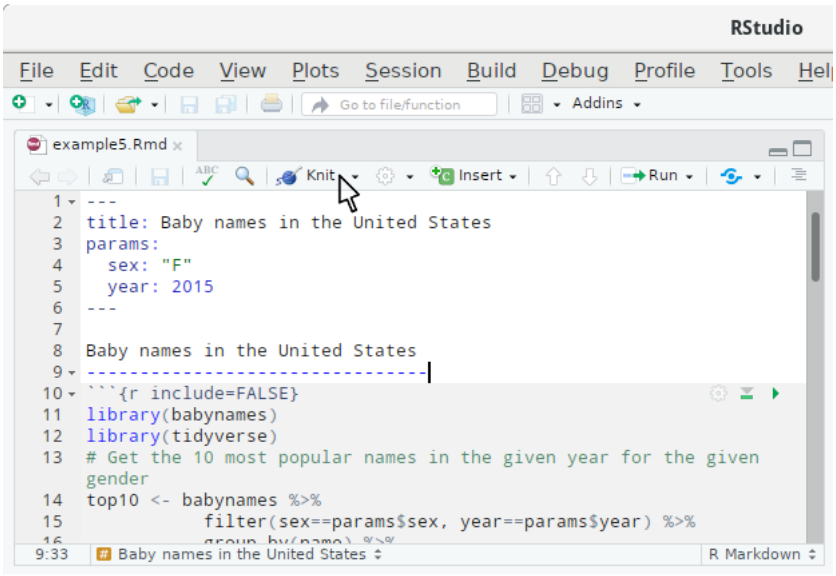


Figure 1: Knit *button in RStudio*

Caching

You do not want to repeat expensive calculations every time you recompile the report. In this case you can use the option `cache=TRUE`. In this case `knitr` will only re-run this chunk if its code changes.

This can however be dangerous. `knitr` will for example *not* detect if a data file you read in has changed. So even after the file has changed, `knitr` will not update the chunk output. The same problem arises if a chunk uses a variable created in another chunk. There is however a solution for the latter problem. You can either let `knitr` figure out automatically how chunks depend on each other (using the option `autodep=TRUE`, which works in most circumstances) or manually set up dependencies (using the option `dependson="label-of-other-chunk"`, which is safer, but also a lot more work). `knitr` can then take the dependencies between chunks into account. The knitr documentation contains examples illustrating the cache functionality in more detail

How to knit

So far we have looked at how to write R Markdown documents, but we have not looked at how we "knit", i.e. how we translate the R Markdown into the final report.

If you open the R Markdown file (extension `.Rmd`) in RStudio you can simply click on the *Knit* button.

You can also compile the document by using

```
library(rmarkdown)
render("filename.Rmd")
```

In RStudio you can also run individual chunks by clicking on the small play icon at the top right of each chunk.

Output formats

R Markdown supports various output formats, the main ones being HTML, PDF (requires a working installation of LaTeX) and Word. When using RStudio you can select the output format by clicking on the expand icon next to the *Knit* button and selecting the corresponding output.

When using the command line you can use the argument `output_format` and set it to `html_document`, `pdf_document` or `word_document` (or other supported formats).

You can also specify the output format and other options (including the title of the report) by including a YAML header. The YAML header is delimited by a line consisting of three dashes ("---").

####[example] YAML header The YAML header below sets the title and author. It also specifies HTML as the default output.

```
---
title: A report written in R Markdown
```

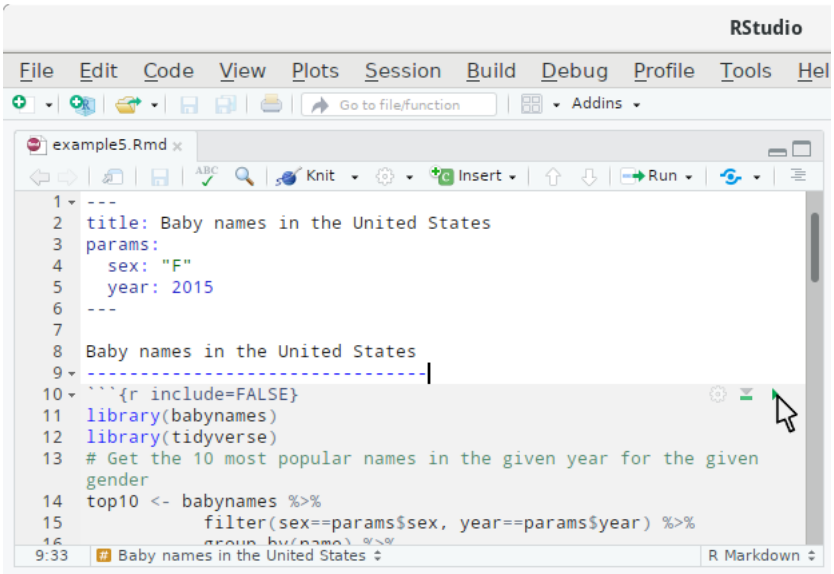


Figure 2: Play *button to run and preview individual chunks*

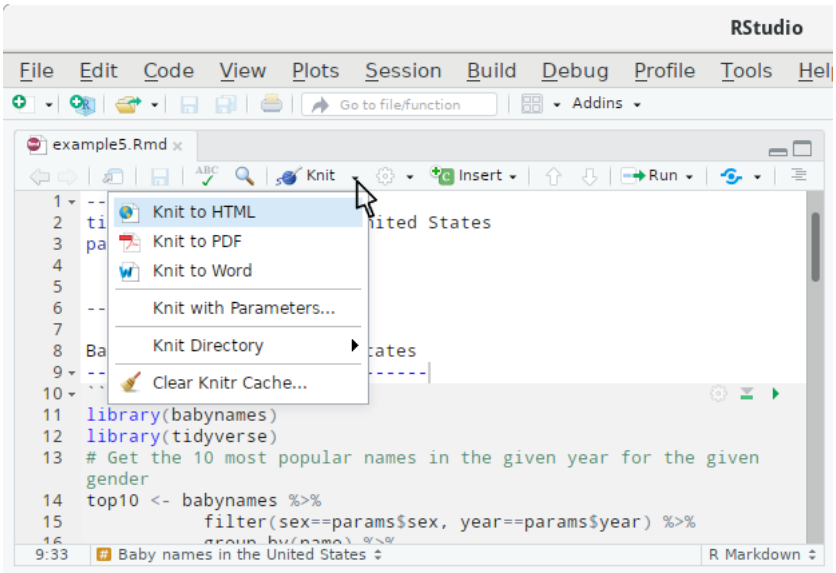


Figure 3: Choosing output format using RStudio

```
author: A N Other
output: html_document
---

# Content

R Markdown content goes here ...
```

####[/example]

Parameterising your report

Suppose you use R Markdown to generate a monthly sales report. You then might want to run it for different months without having to change the R Markdown source.

You can declare parameters in the YAML header using the following syntax.

```
---
title: A Parameterised Report
params:
    key1: value1
    key2: value2
    ...
---
```

These parameters (`key1` and `key2`) are then available in your R code chunks as `params\$key1` and `params\$key2`. Their default values will be `value1` and `value2`.

When using RStudio you can click on *Knit with parameters* which will then show a dialog box letting you input values for the parameters.

You can also set the parameters from the command line using the following syntax.

```
library(rmarkdown)
render("report.Rmd", params=list(key1="anothervalue1", key2="anothervalue2"))
```

Note that you can only set parameters you have declared in the YAML header.

Example 5 (Baby names). The R Markdown document below has two parameters: `sex` and `year`.

```
---
title: Baby names in the United States
params:
  sex: "F"
  year: 2015
---

Baby names in the United States
-------------------------------
```{r include=FALSE}
library(babynames)
library(tidyverse)
# Get the 10 most popular names in the given year for the given gender
top10 <- babynames %>%
           filter(sex==params$sex, year==params$year) %>%
           group_by(name) %>%
           summarize(total = sum(n)) %>%
           arrange(desc(total)) %>%
           slice(1:10)

if(params$sex=="M") {
    boysorgirls <- "boys"
} else {
```
```

```
    boysorgirls <- "girls"
}
```

Top 10 baby names for `r boysorgirls` in `r params$year`

```{r echo=FALSE}
library(knitr)
top10 %>%
    kable()
```

History

```{r echo=FALSE}
# extract popularity of those names over the last 50 years
chartdata <- babynames %>%
                  filter(year>params$year-50, year<=params$year,
                         sex==params$sex,
                         name %in% top10$name) %>%
                  group_by(name, year) %>%
                  summarize(total = sum(n))

# Prevent ggplot from showing the names in alphabetical order
chartdata <- chartdata %>%
                 ungroup() %>%
                 mutate(name=factor(name, levels=top10$name))

# Generate time series plots
ggplot(data=chartdata) +
    geom_line(aes(x=year, y=total)) +
    facet_wrap(~name)
```

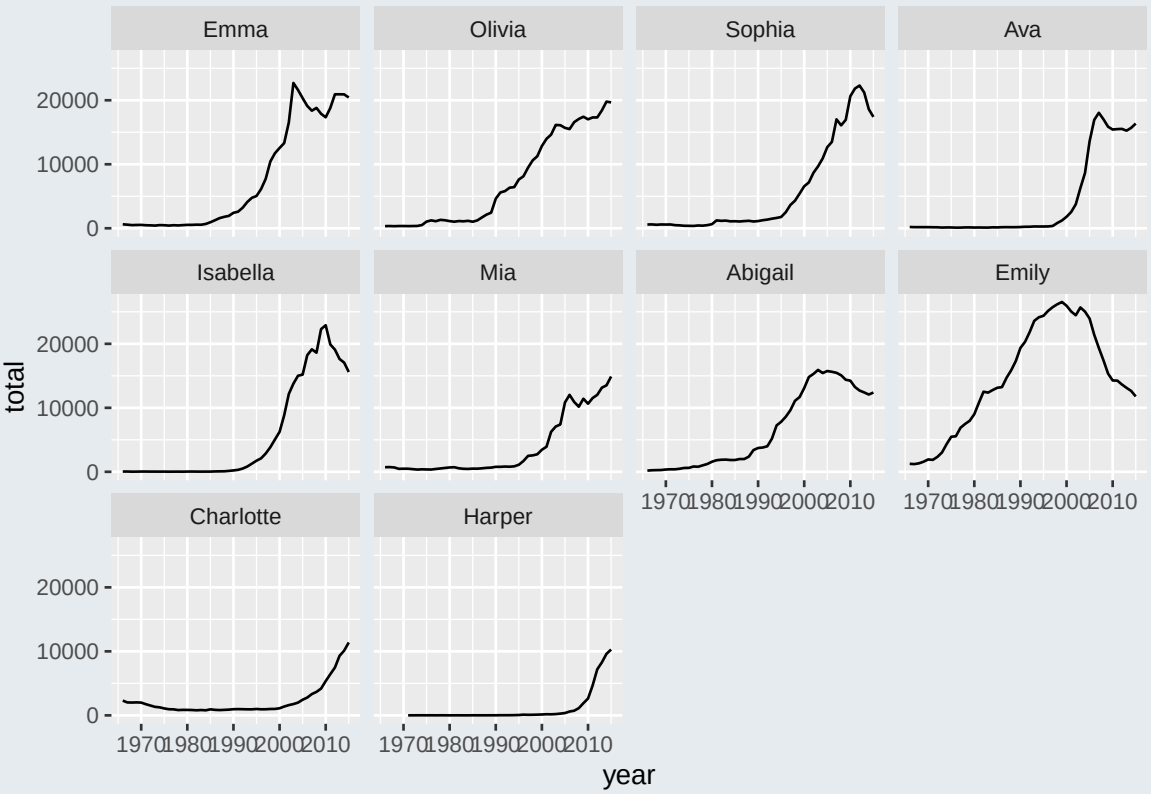
With the default values it produces the output below.

Baby names in the United States

Top 10 baby names for girls in 2015

name	total
Emma	20435
Olivia	19669
Sophia	17402
Ava	16361
Isabella	15594
Mia	14892
Abigail	12390
Emily	11780
Charlotte	11390
Harper	10291

History



We can generate a report about the most popular names for boys in 1977 using

```
library(rmarkdown)
render("example5.Rmd", params=list(sex="M", year=1977))
```

More detailed information (including how to set up Shiny controls for the parameters) is available on the RStudio website.

R Markdown cheat sheet

https://github.com/rstudio/cheatsheets/blob/main/rmarkdown-2.0.pdf

RStudio have put together a very handy and compact cheat sheet for R Markdown.

R Markdown - The Definitive Guide

https://bookdown.org/yihui/rmarkdown/

This book, published by the creaters of R Markdown, contains a very detailed guide of all the features of R Markdown, with some handy hints for commands.

R Markdown tutorial

http://rmarkdown.rstudio.com/lesson-1.html

RStudio have produced a short tutorial for R Markdown.

Background reading: Chapter 27 (and 29 & 30) of R for Data Science

http://r4ds.had.co.nz/r-markdown.html

Chapter 27 of *R for Data Science* gives an introduction to R Markdown. Chapter 29 explains the different output

formats and the associated options. Chapter 30 contains helpful hints for developing an efficient analytics workflow using R Markdown.

Using knitr without Rmarkdown

Markdown has many strengths (ease of use, ability to convert to many output formats, etc.), however one weakness is that Markdown does not give you fine-grain control over detailed aspects of the layout.

If you are familiar with HTML or LaTeX, you can also embed knitr chunks in these documents while retaining fine-grain control of how the content is rendered.

knitr chunks in HTML documents

`knitr` code chunks can be included in HTML using

Inline `knitr` expressions in HTML use the syntax ".

The file containing the HTML with the R chunks should then be saved as a `.Rhtml` file. This file can then be knitted in RStudio by clicking in the *Knit* button. Alternatively, you can run

```
library(knitr)
knit("report.Rhtml")
```

Example 6 (Baby names as Rhtml file).

```
<!DOCTYPE html>
<html>
  <head>
    <title>Baby names in the United States</title>
  </head>
  <body>



<h2>Top 10 baby names for
    in </h2>




<h2>History</h2>


  </body>
</html>

<html>
```

knitr chunks in LaTeX documents (Rnw files)

`knitr` code chunks can be included in LaTeX documents using standard "noweb" syntax.

```
<<#chunk-options>>=
# R code goes here
@
```

Inline `knitr` expressions in LaTeX use the syntax `\Sexpr{#R expression goes here}`. If you are wondering about the name of the command: R is a re-implementation of the S engine and a key concept in the implementation of both R and S is an S expression (`SEXP` in C code).

The file containing the LaTeX code with the R chunks is then saved as `.Rnw` file. The file can then be compiled into a PDF file by clicking on the PDF icon in RStudio. The file can be translated into a LaTeX file using

```
library(knitr)
knit("report.Rnw")
```

You then need to run LaTeX to convert the TeX file into a PDF document.

Example 7 (Baby names as Rnw file).

```
\documentclass{article}
\usepackage{booktabs}

<<echo=FALSE>>=
# Paremeters need to be set manually when not using rmarkdown
params <- list(sex="F", year=2015)
@

\begin{document}
\SweaveOpts{concordance=TRUE}

\title{Baby names in the United States}
\maketitle

<<echo=FALSE>>=
library(babynames)
library(tidyverse)
# Get the 10 most popular names in the given year for the given gender
top10 <- babynames %>%
            filter(sex==params$sex, year==params$year) %>%
            group_by(name) %>%
            summarize(total = sum(n)) %>%
            arrange(desc(total)) %>%
            slice(1:10)
if (params$sex=="M") {
  boysorgirls <- "boys"
} else {
  boysorgirls <- "girls"
}
@

\section*{Top 10 baby names for \Sexpr{boysorgirls} in \Sexpr{params$year}}

<<echo=FALSE, results=tex>>=
library(knitr)
top10 %>%
    kable(format="latex", booktabs=TRUE)
@

\section*{History}

<<echo=FALSE, fig=TRUE>>=
# extract popularity of those names over the last 50 years
chartdata <- babynames %>%
                filter(year>params$year-50, year<=params$year,
                       sex==params$sex,
                       name %in% top10$name) %>%
                group_by(name, year) %>%
                summarize(total = sum(n))

# Prevent ggplot from showing the names in alphabetical order
chartdata <- chartdata %>%
```

```
                ungroup() %>%
                mutate(name=factor(name, levels=top10$name))

# Generate time series plots
ggplot(data=chartdata) +
      geom_line(aes(x=year, y=total)) +
      facet_wrap(~name)
@

\end{document}
```

Sweave

`knitr` was inspired by `Sweave`, the literate programming functionality built into base R. The basic syntax of `knitr` for LaTeX is very similar to the one used by `Sweave`. More information about `Sweave` is available in the Sweave manual.