Code Chunks

Paul Johnson , Director, CRMDA <pauljohn@ku.edu>
September 1, 2017

This is about code chunks and interacting with statistical software.

1 Introduction: Terminology

A code chunk is a runable piece of R code. Producing the document will re-run calculations.

The CRMDA document family, provided by the `crmda` package for R (so far includes):

1. Guides
   - rmd2html-guide
   - rnw2pdf-guide-knit
   - rnw2pdf-guide-sweave
2. Reports
   - rmd2pdf-report
   - rnw2pdf-report-knit
   - rnw2pdf-report-sweave

Code chunks are not a required element in these documents, but the templates and the script processors are set up to allow them.

When a document is prepared in `noweb`/LaTeXor `Rmarkdown`, a document will be converted through several formats to arrive in either HTML or PDF. If everything "just works", then authors might not worry too much about success at each individual stage. In my experience, more elaborate documents almost never "just work". Understanding the sequence of transitions can help in correcting problems.

This document is about one of the earliest stages in document processing. The transition from `Rnw -> tex`, or from `Rmd -> md`, will scan the document for "code chunks", send them to R, and then the results might be put to use in the document. This is called "weaving" or "knitting", depending on which chunk-processing function is used. The older, traditional method is called `Sweave`, while the newer engine is called `knitr`.

1425 Jayhawk Blvd.
470 Watson Library
Lawrence, KS 66045-7594

Phone: 785-864-3353
Website: crmda.ku.edu
Email: crmda@ku.edu

2 Enter "code chunks"

Suppose a professor is writing about the psychology of adolescence or social conflict in Uganda. In the "old" paradigm, the professor will do some statistical analysis in SAS or SPSS and the copy/paste output into the document. This is a tedious, error prone process. Documents produced in this way are difficult to keep up-to-date and difficult to proof read.

Instead of cutting and pasting, we instead insert code chunks that are *run during document preparation*. The advantages of this are obvious. The statistical code and results never become "out of step" with the final document. If the raw data is revised somehow, we no longer repeat the "old" fashioned process of re-running the analysis and then "copy/pasting" the results into a revised document. Instead, we run through the document preparation steps again and the results are *automatically* updated and revised.

Code chunks are allowed in both `noweb`/LaTeX and `Rmarkdown` documents intended for either HTML or PDF. In R, the original method for creating code is called "Sweave". A code chunk is created by a somewhat cumbersome notation that begins with "<<>>=" and ends with `"@"`. Here is a code chunk that runs a regression model.

```
<<>>=
lm(y ~ x, data = dat)
@
```

The chunks in `noweb`/LaTeX documents can be processed by `knitr` as well. In `Rmarkdown` documents, `knitr` is the only chunk-processing technology that is available. In `Rmarkdown` documents, the outer boundaries are changed to three back ticks, along with squiggly braces and the letter "r", which designates the language being processed (`knitr` can handle several languages).

````
```{r}
lm(y ~ x, data = dat)
```
````

These are simple, unnamed chunks. In actual usage, the beginning of the chunk usually has several additional arguments, including a name for the chunk. Names are helpful because error messages will later report the name of the failed chunk. For that reason, we suggest, as a general policy, that *chunks should be named*.

```
<<mychunk10>>=
lm(y ~ x, data = dat)
@
```

````
```{r mychunk10}
lm(y ~ x, data = dat)
```
````

The options, which control if the code is displayed in the document, or if the output is included, and so forth, can be specified after the chunk's name.

3 Illustration of Chunk Features with R and `knitr`

Rather than going through all possible chunk arguments, we now survey (and give examples) of the chunk variations that we require in documents, for any frontend or backend. We need the ability to create chunks that:

1. are not evaluated, but are displayed "beautifully" to the reader, with syntax highlighting.
2. are not displayed, but are evaluated, and the results may (or may not) be displayed for the reader.
3. create graphics, which are automatically included in the document.
4. create graphics (or other files) that are not automatically included in the document. Graphics are saved in image files that can be inserted at a different part in the document
5. import previous chunks for re-analysis.

This document is prepared with `Rmarkdown`. The chunks are converted from `Rmd` to `md` by the `knitr` functions.

Because this document is using the PDF backend, we are able to use the LaTeX `listings` package to display the results. `listings` is a highly customizable framework that can beautify the code and output displays.

The listings display is set to show code chunks in a light gray background with a monospace typewriter font. I prefer to not display the R prompt ">" in the listings display for code input. In addition, contrary to the `Rmarkdown` default, I do not want the R comment symbol, `##`, at the beginning of every line of output.

3.1 Types of chunks we need

1. A chunk that is evaluated, echoed, both input and output. This is a standard chunk, no chunk options are used:

   ```
   ```{r chunk10}
   set.seed(234234)
   x <- rnorm(100)
   mean(x)
   ```
   ```

   The user will see both the input code and the output, each in a separate box:

   ```
   set.seed(234234)
   x <- rnorm(100)
   mean(x)
   ```

   ```
   [1] -0.1004232
   ```

   Notice the code highlighting is not entirely successful, as the function `set.seed` is only half-highlighted.

2. A chunk with commands that are echoed into the document, but not evaluated (`eval=F`).

```
  ```{r chunk30, echo=F}
  set.seed(234234)
  x <- rnorm(100)
  ```
```

The user will not see any code that runs, but only a result box:

```
  ```{r chunk20, eval=F}
  set.seed(234234)
  x <- rnorm(100)
  mean(x)
  ```
```

When the document is compiled, the reader will see the depiction of the code, which is (by default) beautified and reformatted:

```
set.seed(234234)
x <- rnorm(100)
mean(x)
```

3. A chunk that is evaluated, with output displayed, but code is not echoed (`echo=F`). It is not necessary to specify `eval=T` because that is a default.

```
  ```{r chunk30, echo=F}
  set.seed(234234)
  x <- rnorm(100)
  ```
```

The user will not see any code that runs, but only a result box:

```
[1] -0.1004232
```

4. A hidden code chunk. A chunk that is evaluated, but neither is the input nor output displayed (`include=F`)

```
  ```{r chunk40, include=F}
  set.seed(234234)
  x <- rnorm(100)
  mean(x)
  ```
```

Here's what happens when that is processed:

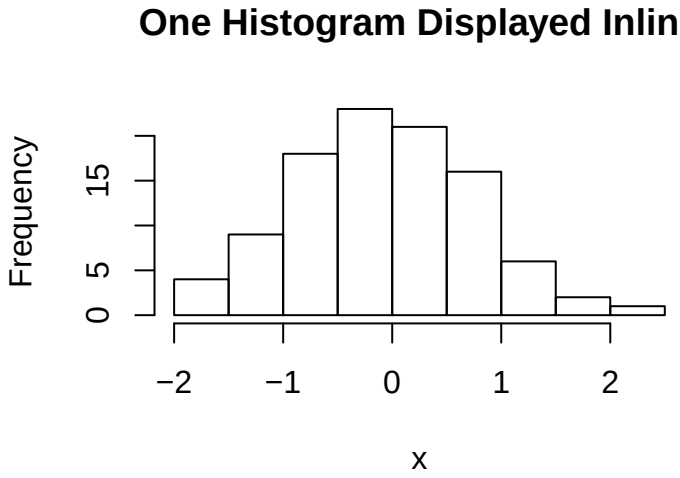
What is the grammatically correct way to say "did you see nothing?" You should not even see an empty box? After that, the object `x` exists in the on-going R session, it can be put to use.

3.2 Chunks with graphics

5. A chunk that creates a graph, and allows it to be inserted into the document, but the code is not echoed.

```
  ```{r chunk50, fig=T, fig.height=3, fig.width=4,
     fig.show="hold", echo=F}
   hist(x, main = "One Histogram Displayed Inline")
  ```
```

The result of that should simply be a graph inserted into the final document, with no signal to the reader that it was produced by a code chunk.



6. Save a graph in a file and display it at another point in the document.

   A. First, I demonstrate feature unique to the PDF backend. PDF documents have features for "floating" tables and figures and these can be used. A chunk that creates a graph and provides a caption will cause a float to be created.

```
  ```{r chunk60, fig=T, fig.height=3, fig.width=4,
     fig.show="hold", echo=F, fig.cap = "\\label{fig:hfloat}A
     Floating Histogram"}
  hist(x, main = "One Histogram")
  ```
```

   Observe in the output we have a numbers, floating figure. Because we inserted a label with the caption, we can cross-reference Figure 1. Figure numbers will be adjusted automatically as new figures are added before and after this chunk. This feature is not available in HTML.

   B. For HTML output, the best we can do is save a graph, but does not allow it to be displayed immediately.

```
  ```{r chunk65, fig=T, fig.height=3, fig.width=4,
     fig.show="hide", dev=c("pdf", "png")}
  hist(x, main = "Another Histogram")
  ```
```

```
hist(x, main = "Another Histogram")
```

   Why do this? In HTML, we don't have the option to create figure floats, but the next-best option is to show code for a figure, but prevent its inclusion in the document by setting `fig.show="hide"`.

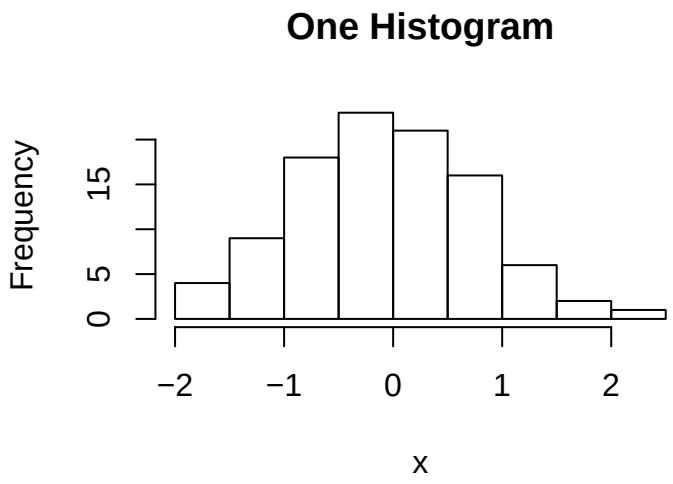


Figure 1: A Floating Histogram

In this code chunk, we asked for output in 2 formats, one as png and one as pdf.

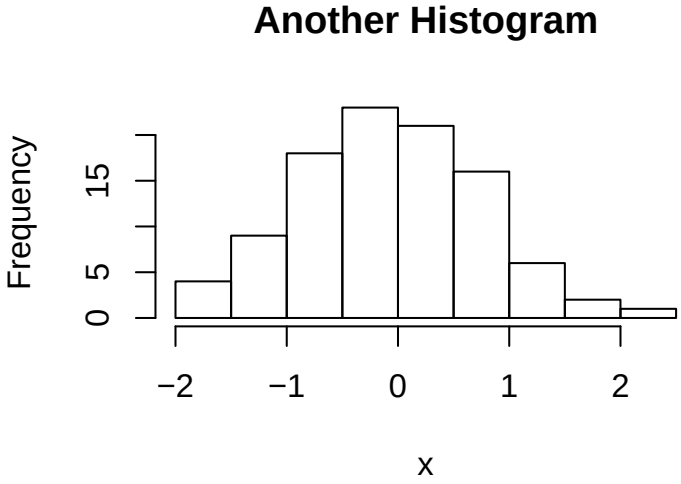
Now that the image file exists, we can insert it manually, because the pdf was saved in a file named "tmpout/p-chunk60-1.pdf". I chose the name "tmpout" in the template, that is configurable. But I like that name pretty much. Here, for example, is an HTML table in which I have embedded that image

If we want to insert the figure with markdown tools, we could. This is the code that would typically be used.

```
![Another Histogram](tmpout/p-chunk65-1.pdf)
```

This is inflexible because the image cannot be rescaled. Also, note that markdown does not have a concept of "centering" or "alignment" of a figure, so to a significant extent we have to focus on tools in the backend document language. Here we suggest using backend-specific code to include–and scale–the figure. This inserts the figure using LaTeX terminology in a PDF document__

```
\includegraphics[width=3.5in]{tmpout/p-chunk65-1.pdf}
```



Here we are happy to see that the *raw LaTeX* code for inserting graphics does work (but do not understand why).

If we are targeting an HTML backend, we would write a similar request in HTML code, using the file `tmpout/p-chunk65-1.png`:

```
<img src="tmpout/p-chunk65-1.png" alt="a floating histogram"
   width="308"
height = "216">
```

Here, we are guessing that the end user's screen will tolerate an image that is 308 pixels wide. If I were writing in HTML at the moment, I'd try harder on sizing that appropriately. But this is a PDF document and this is just an example of what an HTML author might do.

7. One chunk that shows a series of commands. This is an example of a feature in the `knitr` chunk-processing framework. It is not directly accessible from Sweave, but it can be achieved by some careful coding. The `knitr` code chunk option will . It is possible to display the whole graph created by the series of commands.

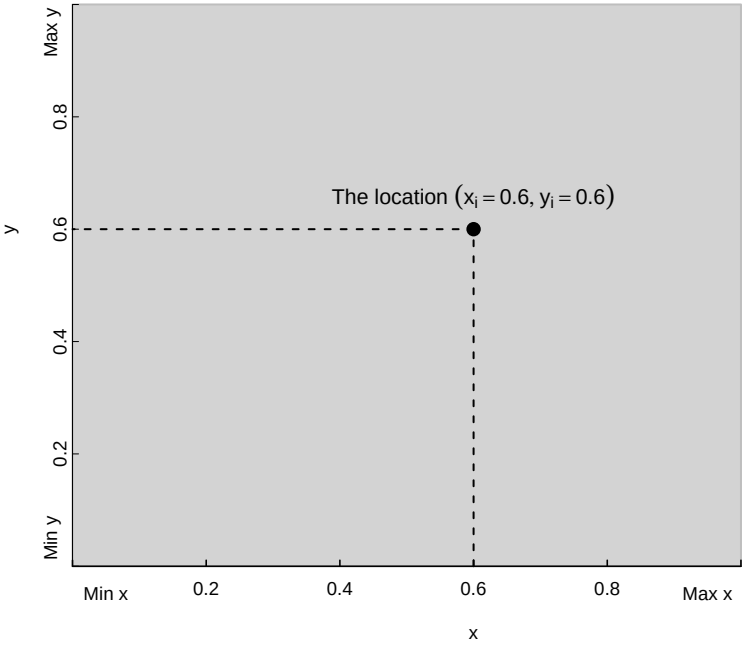
In this case, we demonstrate the usual R plotting exercise in which a "blank" plot is created, and then lines, points, and labels are added one by one.

```
plot(c(0, 1), c(0, 1), xlim = c(0,1), ylim = c(0,1), type = "n",
   ann = FALSE, axes = F)
rect(0, 0, 1, 1, col = "light grey", border = "grey")
axis(1, tck = 0.01, pos = 0, cex.axis = 0.6, padj = -3, lwd = 0.8,
     at = seq(0, 1, by = 0.2), labels = c("", seq(0.2,0.8, by=0.2),
        ""))
axis(2, tck = 0.01, pos = 0, cex.axis = 0.6, padj = 3, lwd = 0.3,
     at = seq(0, 1, by = 0.2), labels = c("", seq(0.2,0.8, by=0.2),
        ""))
mtext(expression(x), side = 1, line = 0.5, at = .6, cex = .6)
mtext(expression(y), side = 2, line = 0.5, at = .6, cex = .6)
mtext(c("Min x", "Max x"), side = 1, line = -0.5, at = c(0.05,
   0.95), cex = .6)
mtext(c("Min y", "Max y"), side = 2, line = -0.5, at = c(0.05,
   0.95), cex = .6)
lines(c(.6, .6, 0), c(0, .6, .6), lty = "dashed")
text(.6, .6, expression(paste("The location ",
                              group("(",list(x[i] == .6, y[i] ==
                                 .6),")"))), pos = 3, cex = .7)
points(.6, .6, pch = 16)
```

I want to run that code from top to bottom, but I don't want to retype it. Both Sweave and knitr allow one to retrieve code from a chunk and put it to use again. This demonstrates the knitr method, which uses the chunk option `ref.label`.

```
```{r chunk75, ref.label='chunk71', echo=F, fig=T, fig.keep="last",
   collapse=T, fig.width=5, fig.height=5}
```
```

This produces one figure, which happens to illustrate the Cartesian plane (between 0 and 1 on both axes):



However, we might be in teaching mode and we need to demonstrate the effect of each successive R function in the process of creating the figure. If one is using `Sweave`, one will have to write many separate chunks, each to accomplish each stage. In comparison, the `knitr` engine has a special option, `fig.keep`, which instructs the system to keep a snapshot of each separate image in the creation of this figure.

```
```{r chunk76, ref.label='chunk71', echo=F, fig=T, include=F,
   fig.keep="all", collapse=T, fig.width=4, fig.height=4}
```
```

A quick check of the tmpout directory shows that this code created several graphs. Observe:

```
list.files("tmpout", pattern="p-chunk76.*pdf")
```

```
 [1] "p-chunk76-1.pdf"  "p-chunk76-10.pdf" "p-chunk76-11.pdf"
 [4] "p-chunk76-2.pdf"  "p-chunk76-3.pdf"  "p-chunk76-4.pdf"
 [7] "p-chunk76-5.pdf"  "p-chunk76-6.pdf"  "p-chunk76-7.pdf"
[10] "p-chunk76-8.pdf"  "p-chunk76-9.pdf"
```

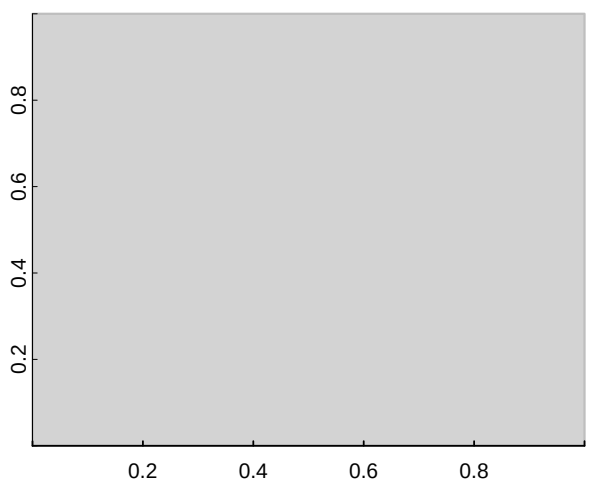
There is a separate output file for each stage in the transition, `p-chunk-76-1.pdf` through `p-chunk-76-11.pdf`. It is a bit tricky to display a matrix of figures "automatically", but I'll use a LaTeX structure to do this work.

3.3 Other special features of `knitr`

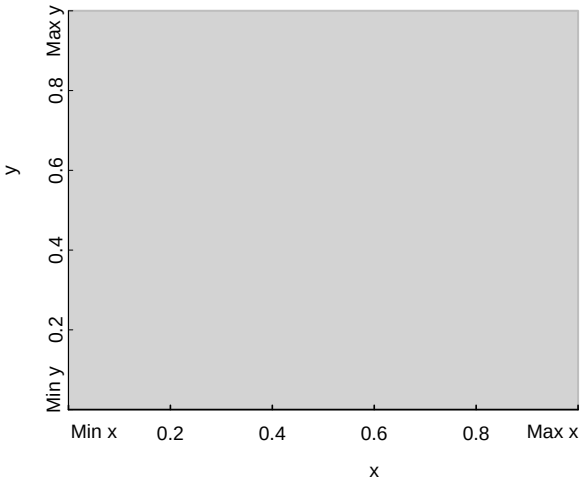
3.3.1 Chunk caching

This is "SLOW!" That's a common complaint about compiling a document that has substantial code chunks. It is SLOW. The delay discourages authors from following my advice to "compile

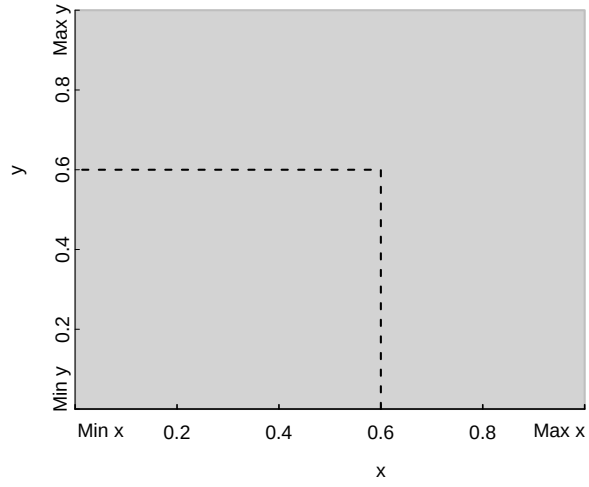
Figure 2: Four snapshots



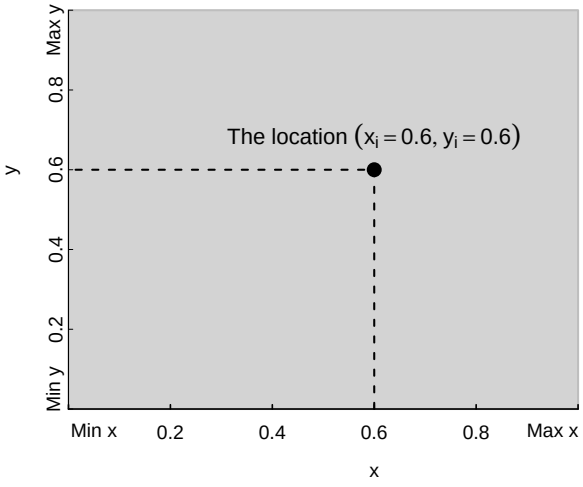
(a) Blank Axes



(b) Labels for Axes



(c) Begin decorations



(d) Finished

early, compile often."

When I am writing in LaTeX, I have found 2 strategies to deal with the slowness caused by processing R chunks. First, I have sometimes created "document branches". If we put all of the R code in a document branch, and then flag that branch as "inactive", then the document can be compiled without re-processing all of the R code. Second, I have used a simpler "two document" strategy. I have one `noweb`/LaTeX
file with code chunks. It has nothing else, no commentary. Using the `split=T` flag, that document can be compiled to run the R code. The run creates separate code files for each of the chunks. Then I have a second document that inputs those chunks where desired. In order to compile the document, then, it is not necessary for me to re-run all of the calculations. Both of these approaches require some planning and it is important to run the code-producing branch (or file) when generating a final version of the document.

Neither of my workaround features are available for Rmarkdown authors, in large part because the `split=T` document option is not allowed in Rmarkdown. In fairness, I admit that the "homemade" document workarounds that I use may be difficult to administer (especially for novices). The `Rmarkdown` and `knitr` authors have employed an alternative strategy known as "code caching". When a chunk is processed, a copy of the calculations can be saved and used next time. The "magic trick", if it works perfectly, will never re-run a chunk unless it is necessary. In the implementation, my experience has been somewhat mixed, but quite a bit of effort has been made by the package authors.

In an Rmarkdown document that reports on extensive simulations, I would almost certainly avoid embedding the code in the document and I'd write a separate document that can conduct calculations and export tables and graphs. This is more difficult in Rmarkdown than it is in PDF documents, but it can be done.

3.4 Document-wide options

3.5 Presentable tables

In a report document, it would be unusual to display a large chunk of R code or output. Instead, reports typically include stylized tables, graphs, and equations.

4 More about Rmarkdown

4.1 Use a text editor

An R markdown file is suffixed ".Rmd", *not* ".rmd". It is a "raw text" file. Don't edit it with MS Word. Instead, use a programmers file editor, such as Emacs, Notepad++, or an integrated development environment, such as Rstudio, to edit the file.

Remember that the markdown philosophy is that a document should look reasonably nice, even before it is compiled. That means authors should take care that their document has reasonably "shaped" paragraphs. Lines should generally be 80 characters or less.

4.2 Additional Material about Rmarkdown

1. *R Markdown Reference Guide*
2. *R Markdown Cheatsheet*
3. Yihui Xie. 2015. *Dynamic Documents with R and Knitr. 2ed* Boca Raton, Florida: Chapman Hall/CRC. http://yihui.name/knitr. On July 7, 2017, Xie made available a new book manuscript, [*bookdown: Authoring Books and Technical Documents with R Markdown*] (https://bookdown.org/yihui/bookdown).

User-friendly overviews and tutorials on Rmarkdown usage are available.

1. *Markdown Basics*. See also the Rmarkdown page at Rstudio corporation, http://rmarkdown.rstudio.com, which offers a "Get Started" Tutorial.
2. Cosma Shalizi, "Using R Markdown for Class Reports", http://www.stat.cmu.edu/~cshalizi/rmarkdown.
3. Wisconsin University Social Science Computing Collaborative. "R for Researchers: R Markdown". http://www.ssc.wisc.edu/sscc/pubs/RFR/RFR_RMarkdown.html
4. Karl Broman, "Knitr with R Markdown". http://kbroman.org/knitr_knutshell/pages/Rmarkdown.html

5 Session Info

```
warnings()
```

```
NULL
```

```
sessionInfo()
```

```
R version 3.4.1 (2017-06-30)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 17.04

Matrix products: default
BLAS: /usr/lib/libblas/libblas.so.3.7.0
LAPACK: /usr/lib/lapack/liblapack.so.3.7.0

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
[7] base

loaded via a namespace (and not attached):
 [1] compiler_3.4.1  backports_1.1.0 magrittr_1.5    rprojroot_1.2
 [5] htmltools_0.3.6 tools_3.4.1     yaml_2.1.14     Rcpp_0.12.11
 [9] stringi_1.1.5   rmarkdown_1.6   highr_0.6       knitr_1.16
[13] stringr_1.2.0   digest_0.6.12   evaluate_0.10
```