R_eproducible LaTeX

Michael Franke
Institute for Cognitive Science
University of Osnabrück

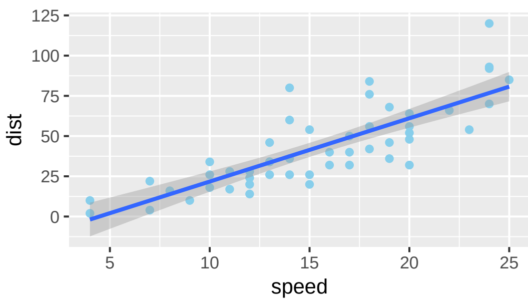
Motivation

Research is reproducible to the extent that it is clear how its results were obtained. Reproducibility of research results is important because it makes it possible for others to learn maximally and efficiently from previous work, so that more energy can be spent on new research questions. It also makes the research process more transparent, helping the scientific process clean itself from natural mistakes or even attempts of fraud. After all scientific progress is mostly a distributed social activity, and not usually the product of a few infallible and saintly geniuses.

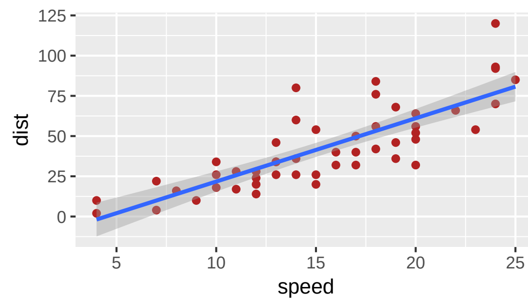
Aiming for reproducibility also helps the researchers themselves during analyses and paper writing. Ideally, results obtained by computational means (e.g., from data analyses or simulation results) should not be copy-pasted into the document, since this is error prone and can create a lot of overhead when the original analyses need to change (e.g., after helpful advice from external reviewers). When working with statistical programming language R, using Rmarkdown is a simple means of achieving this. It allows production of LaTeX-based PDF documents. This shift away from writing papers in pure LaTeX has a number of advantages: Rmarkdown is easier to learn, can be used to create other output formats, such as HTML. On the downside, experienced LaTeX users experience a loss of control. For example, laying out complex subfloats and managing float placement can become difficult, if not impossible. Moreover, nesting of commands (LaTeX command inside an Rmarkdown command inside a LaTeX command) is not generally possible, though often desirable: think of referencing a figure in the caption of another figure, or inserting a cross-reference in a footnote.

The following explains a simple way of compiling fully reproducible papers written directly in LaTeX with computational results stored in CSV-files and read into LaTeX where and when they are needed. This workflow provides an additional burden of bookkeeping during coding and paper writing, but also restores full flexibility. The advantages of this approach are:

- clean separation between coding/data analysis and paper writing (e.g., the possibility to use different editors/editor configurations for each stage)
- immediate extension to other programming languages and mixes of several different programming languages during the coding/data analysis stage
- regains the full flexibility of LaTeX



(a) A plot with blue dots.



(b) A plot with red dots.

Figure 1: A figure with subfigures.

Quick overview

There are usually three types of computational output that show up in a research paper: plots, numbers and (maybe less often) strings. Additionally, numbers and strings are often arranged as tables (e.g., coefficients of a regression analysis). The main idea of R_eproducible LaTeX is to create and store the information needed for the paper in separate files. The advantage is that coding and paper writing are entirely distinct processes, happening in entirely distinct languages and files. Plots are easily saved as PDF-files. Numbers are strings are saved as (appropriately chunked and formatted) CSV-files, which are then read into LaTeX and typeset either inline (using the `csvsimple` package) or as a table (using the `pgfplotstable` package). In order to manipulate number formats in LaTeX, rather than in R (or whatever other programming language), there are convenience functions for normal and scientific notation based on the `siunitx` package. Finally, in order to built the whole paper when something of relevance has changed, we can supply a simple `make` file, as done in this example.

The repository at `https://github.com/michael-franke/R_eproducible_LaTeX` provides this document as a minimal example. It has R code in the file `R_code.r`, which produces plots and numbers which we would like to include in this document.

Plots

Plots are created in the usual way in R, stored as a file and inserted into LaTeX with the usual machinery. This allows fine control over figure placement, labeling etc, also for subfigures and other non-standard constructions. An example is given in Figure 1 which shows two plots generated in the file `R_code.r`.

Numerical results

Number formatting

The `siunitx` package offers a lot of room for typesetting numbers and units. Here, two convenience functions are defined on top of the functionality provided by this package: `\rlnum{}{}` and `\rlnumsci{}{}`. To get a rounded number, we can use `\rlnum{1312.31003}{2}` which produces 1312.31. The second argument gives the number of rounding integers after the comma, so that `\rlnum{1312.31003}{3}` yields 1312.310. We get scientific notation with `\rlnumsci{1312.31003}{3}` to obtain $1.312 \cdot 10^3$.

Single numbers from CSV files

Results from computations in R are stored in CSV files. The `csvsimple` package allows to manipulate data from CSV files in many ways. Here, three simple wrapper functions are defined on top of this package. Each one of the functions:

```
\rlgetvalue{FILENAME.CSV}{COLNAME-KEYS}{KEY}{COLNAME-VALUE}
\rlgetnum{FILENAME.CSV}{COLNAME-KEYS}{KEY}{COLNAME-VALUE}{PRECISION}
\rlgetnumsci{FILENAME.CSV}{COLNAME-KEYS}{KEY}{COLNAME-VALUE}{PRECISION}
```

loads the file `FILENAME.CSV`, finds all rows in which the entry for the column `COLNAME-KEYS` matches the string in `KEY` and returns the value of that row from column `COLNAME-VALUE`. The difference between these three functions is:

- `rlgetvalue` returns what is in the CSV file exactly as it is, be it string or number; all formatting needs to be done in R;
- `rlgetnum` and `rlgetnumsci` expect a numerical entry and pipe it into `\rlnum{}{}` and `\rlnumsci{}{}` respectively; they therefore take an additional `PRECISION` argument.

For example, the data in file `R_data_4_TeX/mystats1.csv` looks like this:

```
variable,mean,sd
dist,42.98,25.769377492025892
speed,15.4,5.2876444352347844
```

A call of `\rlgetnum{mystats1.csv}{variable}{dist}{sd}{2}` gives 25.77.[1]
A call of `\rlgetvalue{R_data_4_TeX/mystats1.csv}{variable}{dist}{sd}` gives 25.769377492025892.

If there are several rows in the CSV file that match the filter, these convenience functions will output a list of all values retrieved. So, if the data rather looked like in file `mystats2.csv`:

[1] Notice that the first argument of all of these functions should specify the full, relative path to the relevant CSV file. However, it is convenient to define specify the path to a folder where all relevant CSV files are found globally using a LaTeX command, as done in this example (see source code and comments therein), so that we do not need to specify the folder name with each retrieval command.

```
variable,stat,value
dist,mean,42.98
speed,mean,15.4
dist,sd,25.769377492025892
speed,sd,5.2876444352347844
```

we get 42.98, 25.77 from `\rlgetnum{mystats2.csv}{variable}{dist}{sd}{2}`.

Tables of numerical results

The `csvsimple` package allows typesetting tables from CSV files, but the `pgfplotstable` package allows for even more flexibility. As an example, we plot the outcome of a regression analysis stored in file `mytable.csv`, which is reproduced here:

```
Rowname,Estimate,Std. Error,t value,Pr(>|t|)
(Intercept),-17.579094890510877,6.758440169379233,-2.601058003022246,0.01231881615380909
speed,3.9324087591240846,0.41551277665712216,9.463989990298366,1.4898364962950983e-12
```

A nice format for this data is in Table 1, which is produced by the code in Figure 2.

	Estimate	SDE	t-value	p-value
(Intercept)	-17.58	$6.76{\cdot}10^{0}$	-2.6	$1.23 \cdot 10^{-2}$
speed	3.93	$4.16{\cdot}10^{-1}$	9.46	$1.49 \cdot 10^{-12}$

Table 1: A table generated from a CSV-file with the code in Figure 2.

Individual unrelated variables

On top of information about numerical variables that are related in an obvious way (and are therefore gathered logically in a data frame), we may want to have a mere list of independent variables, like the number of participants, the average error rate, the mean rating of a post-survey question etc. In the script `R_code.r` we gather these in a list, which is handy because these variables might occur at several places in a script. We finally save this list as a CSV file in long format (each column is a variable, with one row containing the variables' values). It is handy to have the same format and name for this CSV file across different projects, like here: `myvars.csv`.

We can then insert the value of a single variable like `maxSpeed` with command `\rlgetvariable{maxSpeed}`, which produces: 25. This also works for strings: the result of `\rlgetvariable{someString}` is Hello World!

```
\pgfplotstabletypeset[sci zerofill,
col sep = comma,
every head row/.style={before row = \toprule, after row = \midrule},
every last row/.style={after row = \bottomrule},
columns/Rowname/.style={string type, column name={}, column type = l},
columns/Estimate/.style={column name={Estimate}, dec sep align},
columns/Std. Error/.style={column name={SDE}, sci sep align, sci},
columns/t value/.style={column name={$t$-value}, dec sep align},
columns/Pr(>|t|)/.style={column name={$p$-value}, dec sep align }]
{R_data_4_TeX/mytable.csv}
```

Figure 2: Code that produced Table 1.

ToDos

- scientific notation not uniform in Table 1
- OSF show up with `rlgetvariable` but not with the other techniques