Sensible Report Updates

John C. Nash, Telfer School of Management, University of Ottawa

September 2012

Abstract

Many people need to generate reports more than once. Monthly sales reports, for example, or evolving graphs of measurements and statistics. Sometimes it is useful to be able to include fresh calculations. This need has been part of the motivation of the Microsoft OLE framework and similar ventures, but the complexity of such approaches is unattractive. Worse, they only work on one platform and collaboration may be awkward.

The R project for statistical computing developed a rather different and more flexible approach. Under the rubric "Reproducible Research", several tools have been developed, originally so important studies (clinical trials of drugs, for example) could be sure the resulting report contained exactly the results it should, and could be revised properly as new data arrived. These ideas have applicability much wider than statistical computing, and ultimately much wider than the scope of R. Moreover, the tools are cross-platform, very powerful, and can be used collaboratively.

The speaker will show some examples to illustrate how reports can be prepared that automatically update when their content and data are changed. In doing so, he will present a few introductory ideas about R that may be useful to and usable by a general audience.

One last point: R is open source software and the "cost" is a download of approximately 50 MB for the base system. There are, however, more than 3500 add in packages for a huge variety of tasks.

1 Motivations

Many people need to prepare monthly reports in a similar format but with new data. Nevertheless, I see persistent copy-and-paste work going on, with attendant disasters. This is not new. In 1978, a colleague at Agriculture Canada who was the beef marketing specialist sent a tape (remember those big reels of 1/2 inch tape?) to Statistics Canada with the 1978 table of production figures. A copy and paste operation resulted in a recall and pulping of many hundreds of printed reports because the figures published used the 1977 figures.

Humans have a short attention span. Mouse or pointer operations are fine when we only do one or two. They are an onerous chore when we do 50 to 100, and essentially certain to have errors beyond that. (There are studies on spreadsheet correctness under the authorship of Ray Panko of the University of Hawaii for those interested.)

The way round this is to make computers do the dirty and boring work for us. This is usually called **scripting**, since **programming** can scare folk. However, it is a form of programming, though it should not be scary or difficult. Building spreadsheets is a similar form of user-based programming, and people are doing lots of that – often badly of course.

A second motivation is collaboration. Major reports – and even pretty minor ones – rarely have just one author. This results in lots of file attachments being emailed. Inevitably, we have the follow-up messages:

- "Your file won't open on my machine"
- "I can't see anything but gobbledygook in my special-gold-plated editing system"

- "Which version are we talking about?"
- "Why are you not using a proper system like (MacOS / Windows 7 Idiots Edition / SuperLinux4GeeksOnly / FacebookTellTheCompetitionEverything)?"

Or (for uOttawa types like me)

- "My email system truncated the file at 2 MB"

Sidebar: Every September, some flunky in the senior administration gets it into their head that an "important message" needs to be sent to every employee and student of the University. They make sure that a good quality – large file size – image is part of the message, and attach it in Word format, using the most recent version of Office. The resulting 1.5MB message is then dutifully "sent" to about 45000 addresses, and the network slows to a crawl or collapses. Moreover, the recipients usually can't read the message because they don't have the requisite software version. A simple text message will go through fine, and it can include a link to fancy stuff on a server.

In the rest of this article and related talk, I will present a set of tools I have found useful to address the issues raised. As my audience is presumably people using **personal computers**, I will note here that I fully expect people to select and adapt the ideas to their own needs, or perhaps simply file them away for future reference.

2 Some attempts to address wishlist

Computer systems developers have been aware of the issues for some time. Microsoft introduced OLE (Object Linking and Embedding) and evolved it through COM (Component Object Model) towards their .NET infrastructure. These are, however, directed either exclusively or primarily to Microsoft-based platforms. Furthermore, there are complexities that I prefer to avoid, with the involvement of many files and settings, and questions of which applications are compliant with the tools.

For a while there were attempts to build alternatives, such as the OpenDoc structure. This was abandoned, with Apple giving up the trademark finally in 2005. While Apple makes some very nice products, I have a difficult time forgetting Apple 3, Lisa, and Newton on the hardware side, with several ventures in software that have been similarly "dumped".

From my perspective, a more promising line of development comes from the "Literate Programming" initiatives of Donald Knuth and others, starting in 1984, followed up by the "Reproducible Research" thrust that I date to about a decade ago. There are a number of workers from different areas of activity who have contributed. I am most familiar with contributions of Frank Harrell and Friedrich Leisch in the R community, where the importance of getting correctly matched data and calculations has already been mentioned.

3 Reproducible Research

The desiderata we will aim to satisfy under the "reproducible research" banner are as follows.

- A single file or archive should contain all the material to generate the report. This will rarely be perfect, of course, as all calculations are carried out within some sort of computing environment, and controlling such environments can be tricky. Nevertheless, we will aim towards this goal.
- With a particular version of the master file, we should get a particular "published" report. Hence: **reproducible** output.
- We would like our tools that accomplish these goals to be fast, reliable, and efficient in memory, disk, and computing time.
- If possible, we want the whole process, if not the details, to be comprehensible to mere mortals.

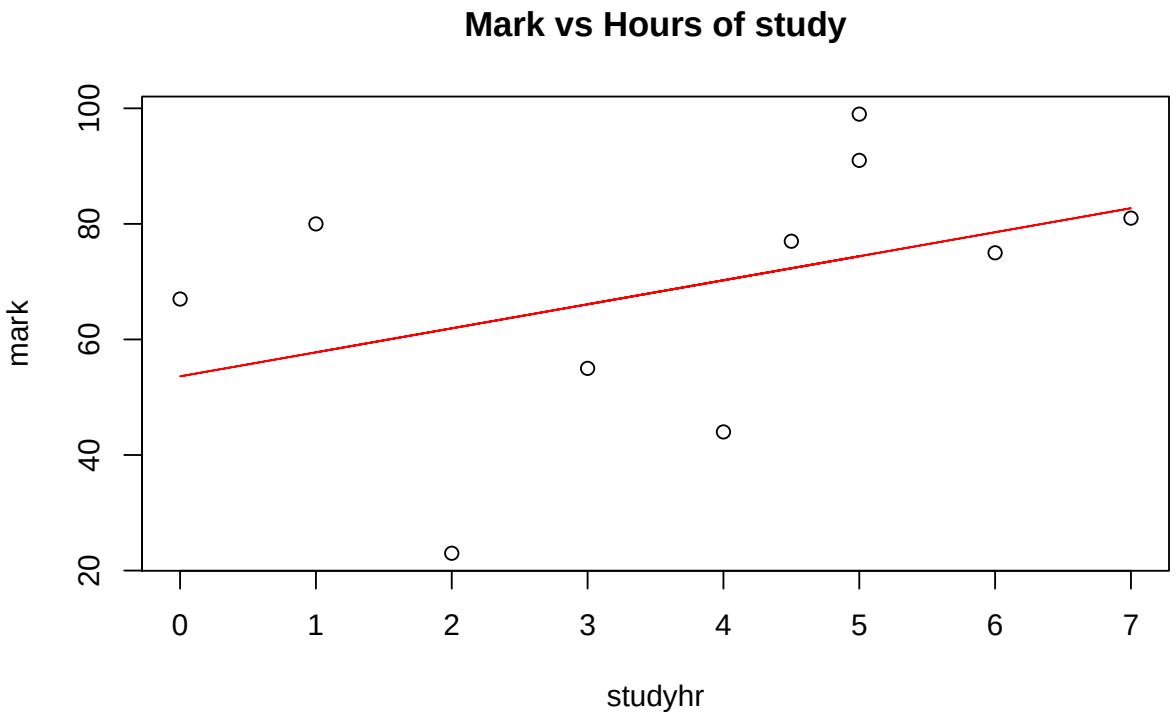
4 A simple example

Let us pretend to be an academic administrator looking at the effect of study time on marks. Here is a set of data and a graph. This uses R , but the reader should not try to do more than follow the generalities of these few lines of script.

```
cat("This is a small program to check marks \\n")

## This is a small program to check marks \n

names <- c("AA", "BB", "CC", "JJ", "NN", "Joe", "Mary", "Bill", "Joan", "Ann")
studyhr <- c(1, 5, 4.5, 4, 3, 6, 7, 2, 0, 5)
gender <- c(0, 0, 1, 1, 0, 0, 1, 0, 1, 1)
mark <- c(80, 91, 77, 44, 55, 75, 81, 23, 67, 99)
plot(studyhr, mark)
title("Mark vs Hours of study")
linmod <- lm(mark ~ studyhr)
points(studyhr, predict(linmod), type = "l", col = "red")
```



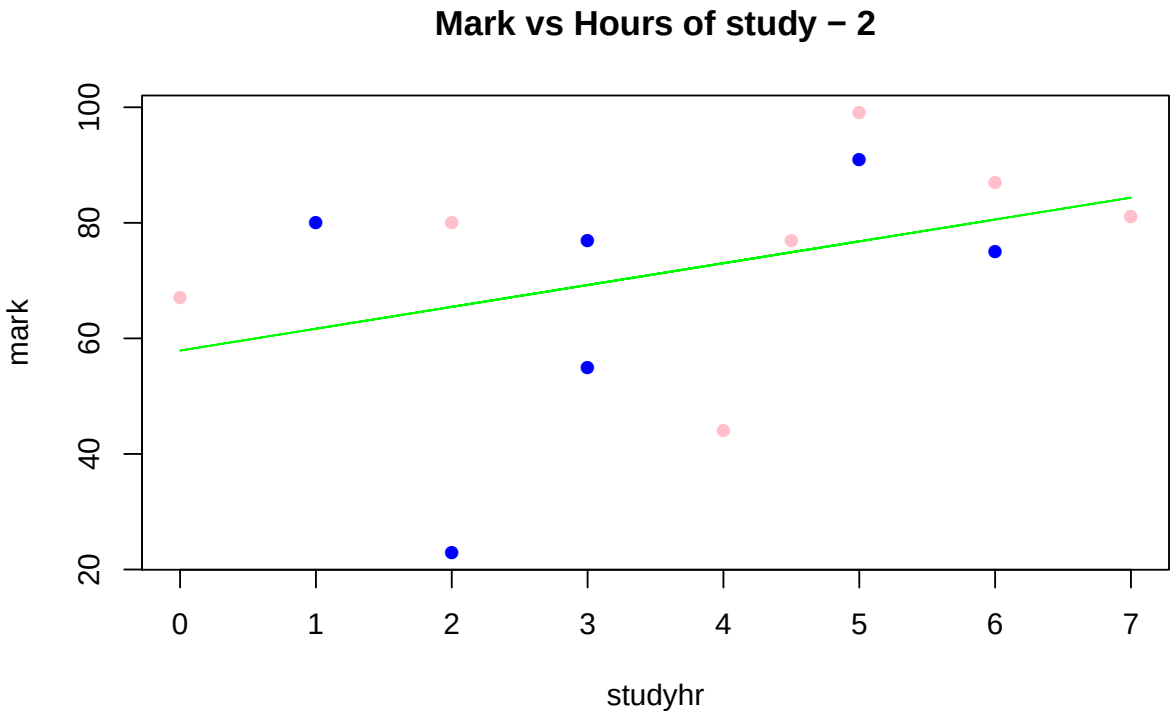
We have our graph, but now someone brings us some more data. And they insist that they need to be able to know if there is a gender difference. Worse, one of the students has not answered how much time they spent studying.

```
cat("Now we get some marks from another group\n")

## Now we get some marks from another group

names <- c("Anne2", "Stephen", "John", "Esther")
gender2 <- c(1, 0, 0, 1)
studyhr2 <- c(2, NA, 3, 6)
mark2 <- c(80, 59, 77, 87)
studyhr <- c(studyhr, studyhr2)
```

```
mark <- c(mark, mark2)
ok <- !is.na(studyhr)
linmod <- lm(mark ~ studyhr)
gender <- c(gender, gender2)
ok <- !is.na(studyhr)
X11()
plot(studyhr, mark, col = "white")
title("Mark vs Hours of study - 2")
linmod <- lm(mark ~ studyhr)
points(studyhr[ok], predict(linmod), type = "l", col = "green")
points(studyhr[gender == 1], mark[gender == 1], col = "pink", pch = 16)
points(studyhr[gender == 0], mark[gender == 0], col = "blue", pch = 16)
```



5 How can this be done?

The general ideas behind the output above are quite straightforward:

- The master document is ”marked up” to include ”code chunks”.
- A pre-processor extracts the code chunks.
- The code chunks are executed by the appropriate system (in this case R , but generalizations are clearly possible).
- The input and output of the code chunks is formatted and inserted into a working document.
- The working document is treated as a normal document for rendering to output such as PDF or printed paper.

The workflow here is relatively simple, though the details can be exceedingly messy. Fortunately, users rarely have to deal with such messiness. There are tools available to do the dirty work. In fact, R has what is called a **Task View** that lists a number of approaches that relate to R at `http://cran.r-project.org/web/views/ReproducibleResearch.html`. I am sure that there are many other approaches and tools, both proprietary and open-source. However, the time and effort to learn, test and document such tools is exceedingly demanding. I keep an eye open for developments, but stick to what works.

What "works" for me are the tools **knitR**, **Sweave** and **odfWeave**. The first two of these are very similar, with knitR being newer and more flexible. (I sometimes need to use Sweave to collaborate with others.) Both process LaTEX documents. odfWeave processes OpenOffice / LibreOffice writer documents. I have found Microsoft Word can edit such documents too, but I have not thoroughly tested that option, and have had some minor difficulties with configuring Windows to run odfWeave fully. I welcome collaboration to develop better understanding and documentation of how to do that properly, and indeed, how to run odfWeave, knitR and Sweave under Windows and Macintosh. Though I am a Linux user, I want others to be able to use what is familiar to them when I wish to collaborate with them.

This article is being prepared with knitR. I will make the source as well as the output PDF documents available. Truthfully, learning LaTEX is a big commitment. I learned it because the payoff in preparing several full-length books was huge. For someone who does a few documents of 1-2 pages, I think odfWeave makes more sense unless there is a TEX guru at hand to prepare the main master document.

6 What is R ?

R started in 1992 as demo/toy of **S** which was developed at Bell Laboratories in the 1970s as a programming language for statistics. **S** was very expensive, as is most proprietary scientific computing software, and beyond the budget of the Robert Gentleman and Ross Ihaka at the University of Auckland. However, they wanted their students to see the possibilities of a computing tool that provided a programming language for statistical computation with very high level commands, e.g., lm() for "linear model". We have seen this in the example above.

It turned out that the "toy" was quite capable, and features were quickly added. In fact, they grew so capable that in 2000 several core **S** team members joined R .

My view is that R succeeds because it has a relatively small backbone (base or core) but adds features via well-defined packages. There are now more than 3500 of these in the Comprehensive R Archive Network repository, and a further 500 or so in a companion set called Bioconductor.

The very strong package structure, insists that all packages be documented and tested before loading on the repository. The structure encourages **vignettes**, which are more extended, tutorial-style, documentation. Examples are not quite mandatory (but should be, in my opinion). Tests are encouraged. A goal to rerun package build checks every 24 hours is a challenge that is sometimes met. Packages that fail are soon withdrawn from the collection – possibly the most stringent of any software repository policy.

There is now a large R community, estimated at 2 million users. There is extensive contributed documentation, in addition to papers, the R Journal, a number of general and specialized help lists, a search engine, several developmental repositories, and many books. The CRAN repository is widely mirrored (5 in Canada).

6.1 Some R features

For the most part, we can consider R to be an interpreted language that has objects. The output of the `lm()` in our example above is a "linear model" object. This has a lot of detail that users mostly can ignore. However, some key consequences of having such an object is that we can `predict()` the model output at the supplied data for variable 'studyhr'. Moreover, there is a print method to provide a nice summary of the object. Want more detail? Frankly, I try to stay away from delving into the nuts and bolts except where I have to.

For casual users, it is more important to know that R almost always can handle whatever data is provided. The way this is done may be messy and awkward in some cases when someone has provided one or more especially strange files. However, for data in common databases there are many tools for accessing information directly. For tables and spreadsheet files, particularly in CSV form, R makes things very easy.

If you looked at the example, you will see that very nice graphs were produced with very few commands. This allows us to maintain a common style. Generally the developers of R and its graphical tools have done an exceptional job of providing powerful and flexible tools. If, of course, you want a whole lot of bells and whistles, be prepared to tinker, but once you get things as you want them, the scripts are relatively immune to mouse fumbles or a change in Office-suite version that can mess up spreadsheet graphics.

Above we used R commands directly, but many users are more familiar with GUIs. R has several for different types of users. It even has several tools for building specialized GUIs, such as `Rpanel`. For general use, I usually suggest the R Commander (abbreviated Rcmdr) interface developed by John Fox of McMaster University. This will seem "clunky" to most Windows/Mac users, but it does the job. Moreover, it automatically provides the commands used to generate the calculations so that they can be later used in scripts and documents. That is really helpful, since it saves users the effort of learning syntax, and computer syntax is always arcane and difficult to remember.

6.2 Should you learn R ?

Probably not! It is likely worth remembering that if you have a report that is based on spreadsheets, and it needs repeated processing to a report, then R can almost certainly do a better job of the calculations and do so in a way that vastly reduces the "busy work" to get the report out. Even if you have such a task, you likely do not need to truly learn R . You may be able to adapt existing examples, including the one above, or in a worst-case, hire someone for a few hours to get the script written (and documented so you understand it!).

For the record, I rarely write R scripts from scratch. I generally cheat and edit one or more that are already available. I'll also admin freely that I expect tools very similar to knitR and friends to become available for other computational tools. I'll cheerfully beg, borrow and steal the details.

7 Collaborating to produce reports

Reports that have to be updated very commonly require two or more people to work together to produce them. If these only require text and static graphics, then there are a number of tools to allow this to happen. The "Changes" features of Office suites have a role to play here, but there remains the issue of how to name and access files so there is not confusion over versions. One approach is to use a common master file on a server. This is the essence of GoogleDocs, but the idea is not new. Indeed, I am associated with an effort from 10 years ago called TellTable (Nash, Adler, and Smith 2004), (Adler, Nash, and Noël 2006). This was, in its time, rather more capable than GoogleDocs in that we had versioning (via the now-dated CVS – Concurrent Versioning System), as well as some tools for analysis of changes. Indeed we could audit who changed what and when in spreadsheets. Because the system was built on many components, all open source, it required a lot of effort to maintain, a task professors are not paid to do, and which does not yield academic rewards either.

The use of versioning, however, makes collaboration a great deal more reliable. There are many tools, both commercial and open source. I regularly use **SubVersion** (`http://subversion.apache.org`) and have also tried **Git** (`http://git-scm.com`). Truthfully, these tend to be rather geeky tools, but for simple collaborations, there are reasonable GUI tools.

There are fierce arguments why a particular tool is better than others, but for the authors of reports that are updated regularly, we only need to worry about the following features:

- How to set up the initial repository.
- How a collaborator gets his initial local copy of the files.

- How local files are updated.
- How an edited version is committed back to the repository.
- How to examine "old" versions of the files.

Of these tasks, the first two are done just once, so one can get help. The update and commit tasks are the ones we need to do regularly. I rarely have to check "old" versions. However, I have learned to rename my resulting pdf files to include the date. That is, a particular version of this article might be ReportUpdates120822.pdf. While operating systems do provide file time-stamps, they are often changed by copy, move and archive operations, and especially by extraction of attachments from emails. Of course, we shoud be using a versioning system!

8 Conclusions

This article is about the decisions that should be made when you need to prepare reports that must be updated from time to time, especially when you need to work with others. When reports are very simple and don't have to be revised as new data comes in, the way in which the report is built has little importance. When the outcome must be correct as new data comes in and the report style should be preserved, we need to consider our options.

Here I have suggested one set of tools that "work". They are cross-platform and open source. Are they perfect? Of course not. But they work quite well, with many users who are often willing to provide advice and help. Moreover, they are being improved continuously.

Perhaps surprisingly, these tools run counter to the contemporary focus on software driven by touch panels and pointer device interfaces. When we must make our work reproducible, it needs to be planned carefully and run by a script. Indeed, it is "hands off", because we humans tend to be clumsy and easily distracted. We can, however, make our computers do much of the work for us.

References

Adler, A., J. C. Nash, and S. Noël (2006, July). Evaluating and implementing a collaborative office document system. *Interacting with Computers 18*(4), 665–682.

Nash, J. C., A. Adler, and N. Smith (2004, July). TellTable Spreadsheet Audit: from technical possibility to operating prototype. In P. Cleary and D. Ward (Eds.), *Proceedings of the 2004 Conference of the European Spreadsheet Risks Interest Group*, pp. 45–56.