A Knitr Demo

Charles J. Geyer

December 12, 2022

1 Licence

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License `http://creativecommons.org/licenses/by-sa/4.0/`.

2 R

The version of R used to make this document is 4.2.1. The version of the `knitr` package used to make this document is 1.40. The version of the `xtable` package used to make this document is 1.8.4.

3 Introduction

This is a demo for using the R package `knitr`. To get started make a regular LaTeX file (like this one) but give it the suffix `.Rnw` instead of `.tex` and then turn it into a PDF file (`bar.pdf`) with the (unix) commands

```
R CMD Sweave --pdf bar.Rnw
```

(as always with LaTeX one may have to rerun this command multiple times until it stops complaining about needing to be rerun). For this to work R needs to have package `knitr` installed.

This works because of the comment

```
%\VignetteEngine{knitr::knitr}
```

(otherwise `R CMD Sweave` would expect `Sweave` rather than `knitr`).

Now include R in our document. Here's a simple example

```
2 + 2

## [1] 4
```

This is not LaTeX. It is a "code chunk" processed by `knitr`. When `knitr` hits such a thing, it processes it, runs R to get the results, and stuffs (by default) the output in the LaTeX file it is creating. The LaTeX between code chunks is copied verbatim (except for `Sexpr`, about which see below). Hence to create an Rnw document you just write plain old LaTeX interspersed with “code chunks” which are plain old R.

4 Plots

4.1 Make Up Data

Plots get a little more complicated. First we make something to plot (simulate regression data).

```
n <- 50
x <- seq(1, n)
a.true <- 3
b.true <- 1.5
y.true <- a.true + b.true * x
s.true <- 17.3
y <- y.true + s.true * rnorm(n)
out1 <- lm(y ~ x)
summary(out1)

##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -29.8660  -9.0664   0.0438   9.1315  29.4561
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3.4527     4.1489  -0.832    0.409
## x             1.6377     0.1416  11.566 1.76e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.45 on 48 degrees of freedom
## Multiple R-squared:  0.7359,Adjusted R-squared:  0.7304
## F-statistic: 133.8 on 1 and 48 DF,  p-value: 1.755e-15
```

4.2 Figure with Code to Make It Shown

Figure 1 (p. 4) is produced by the following code

```r
plot(x, y)
abline(out1)
```

Note that `x`, `y`, and `out1` are remembered from the preceding code chunk. We don't have to regenerate them. All code chunks are part of one R "session".

This was a little tricky. We did this with two code chunks, one visible and one invisible. See the source file `bar.Rnw` to see how it is done. The first code chunk just shows the code (so we can see it above). It has the optional argument `eval=FALSE` which makes it not actually do anything. Then the second code chunk (which we do not see) actually makes the plot. It has optional argument `echo=FALSE` which says it should not be shown (because it would be shown in the figure, which is not what we want).

The second code chunk does not repeat the code of the first chunk. Instead it "quotes" the first chunk, just reuses its code. This follows the DRY/SPOT rule (*don't repeat yourself* or *single point of truth*) so we only have one bit of code for generating the plot. What the reader sees is guaranteed to be the code that made the plot. If we had used cut-and-paste, just repeating the code, the duplicated code might get out of sync after edits.

So making a figure is a bit more complicated in some ways but much simpler in others. Note the following virtues

- The figure is guaranteed to be the one described by the text (at least by the R in the text).
- No messing around with sizing or rotations. It just works!

4.3 Figure with Code to Make It Not Shown

For this example we do a cubic regression on the same data.

```r
out3 <- lm(y ~ x + I(x^2) + I(x^3))
summary(out3)

##
## Call:
## lm(formula = y ~ x + I(x^2) + I(x^3))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -26.723  -9.154  -2.360   9.727  27.868
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.2087859  8.8910666   0.586    0.561
## x           -0.0660437  1.4948825  -0.044    0.965
## I(x^2)       0.0753739  0.0677486   1.113    0.272
```

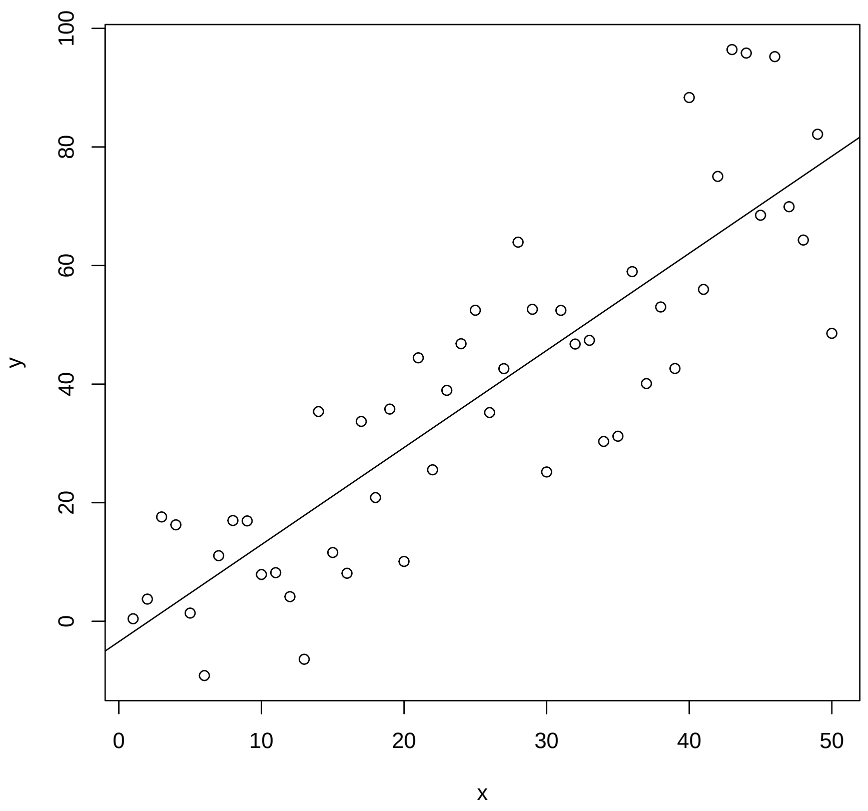

Figure 1: Scatter Plot with Regression Line

```
## I(x^3)      -0.0009204  0.0008737  -1.053    0.298
##
## Residual standard error: 14.55 on 46 degrees of freedom
## Multiple R-squared:  0.7432,Adjusted R-squared:  0.7265
## F-statistic: 44.38 on 3 and 46 DF,  p-value: 1.254e-13
```

If you don't care to show the R code to make a figure, it is simpler still. Figure 2 (p. 6) shows the plot of the (cubic) regression function for these data. It is made by one code chunk with optional argument `echo=FALSE` so it makes a figure and does not show the code.

Also note that every time we rerun `knitr` Figures 1 and 2 change because the simulated data are random. Everything just works. This should tell you the main virtue of `knitr`. It's always correct. There is never a problem with stale cut-and-paste.

5 R in Text

This section illustrates the command `\Sexpr`, which allows running text to be computed by R.

We show some numbers calculated by R interspersed with text. The quadratic and cubic regression coefficients in the preceding regression (page 5) were $\beta_2 = 0.0754$ and $\beta_3 = -0.0009$. Magic!

In order for your document to be truly reproducible, you must never cut-and-paste anything R computes. Always have R recompute it every time the document is processed, either in a code chunk or with `Sexpr`.

6 Tables

The same goes for tables. The `xtable` command is used to make tables. Here is a "table" of sorts in some R printout.

```
out2 <- lm(y ~ x + I(x^2))
anova(out1, out2, out3)

## Analysis of Variance Table
##
## Model 1: y ~ x
## Model 2: y ~ x + I(x^2)
## Model 3: y ~ x + I(x^2) + I(x^3)
##   Res.Df     RSS Df Sum of Sq      F Pr(>F)
## 1     48 10021.1
## 2     47  9978.4  1    42.648 0.2013 0.6557
## 3     46  9743.4  1   235.057 1.1097 0.2976
```

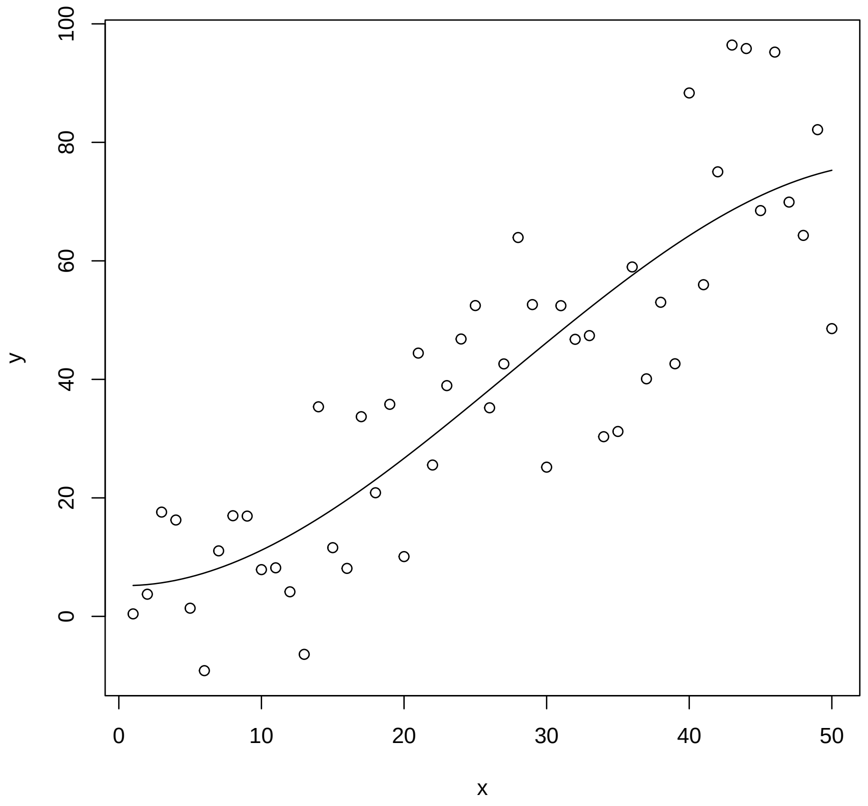

Figure 2: Scatter Plot with Cubic Regression Curve

Table 1: ANOVA Table

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	48	10021.06				
2	47	9978.41	1	42.65	0.201	0.656
3	46	9743.36	1	235.06	1.110	0.298

We want to turn that into a LaTeX table. First we have to figure out what the output of the R function `anova` is and capture it so we can use it.

```
foo <- anova(out1, out2, out3)
class(foo)

## [1] "anova"      "data.frame"
```

So now we are ready to turn the data frame `foo` into Table 1 using the R functions `xtable` and `print.xtable` (that is, the `xtable` method of the generic function `print`), which are found in the R package `xtable`, which is a CRAN package.

The code chunk that makes Table 1 has optional arguments `echo=FALSE` and `results=tex` so we do not see it and the result is not put back in the code chunk but it just treated as ordinary LaTeX.

The reason why we had to use both `xtable` and `print.xtable` is that we needed to use arguments for both. We used the `caption`, `label`, and `digits` arguments of the former and the `table.placement` and `caption.placement` arguments of the latter.

7 Reusing Code Chunks

Code chunks can quote other code chunks. Doing this is an example of following the DRY/SPOT rule (Wikipedia articles Don't Repeat Yourself and Single Point of Truth.

It has already been illustrated above in the section about plotting figures and showing the code in two different code chunks, but it can also be used with any code chunks

Make some data

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

and then do something with it

```
summary(x)

##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -2.966395 -0.696331 -0.005051  0.077783  0.829343  2.245048
```

and then make some other data

```
x <- rnorm(50)
```

and then do the same thing again following the DRY/SPOT rule

```
summary(x)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.55257 -0.53604  0.05722  0.09996  0.87813  2.02935
```

8 Caching Code Chunks

When code chunks take a long time to run, the option `cache=TRUE` can be added to them. Then they are only run once, the results are saved (in a directory, also called folder, on your computer) and every other time the code chunk is processed the cached results are used (so no computer time is taken redoing the long calculation).

If the code in the cached code chunk is changed, then it will be rerun.

But caching is dangerous because it does not know when it should rerun the code when something else has changed. So some warnings.

Never cache a code chunk that has important global side effects. In particular, all calls to R function `library` should not be in cached code chunks. They need to be executed every time `knitr` runs. So they should go in a different code chunk from any code chunks you want to cache.

You can use the `dependson` argument to tell a cached code chunk what other code chunks it depends on. Then the code chunk is rerun whenever what is computed in those "depends on" code chunks change.

The value of the `dependson` chunk option, like all knitr chunk options, is an R object, thus it has to be valid R syntax. The chunk names are character strings. If there are more than one of them, they must be collected into a vector using R function `c`. For example, in `cache=TRUE` the value `TRUE` is the R logical constant `TRUE`. Thus it is not quoted. In `dependson="try1"` the value `"try1"` is the name of a code chunk, so it is a character string. In `dependson=c("try1", "try2", "try3")` the value `c("try1", "try2", "try3")` is a vector of character strings, the vector being made using R function `c`.

For a complete example using `cache` and `dependson` see the notes on Markov chain Monte Carlo, which are in Rmarkdown rather than knitr but Rmarkdown is built on top of knitr and the chunk options are the same for both. The code chunks and their options are seen in the Rmarkdown source for that.

9 Summary

`knitr` is terrific, so important that we cannot get along without it or its older competitor `Sweave` or its newer competitor `Rmarkdown`.

Its virtues are

- The numbers and graphics you report are actually what they are claimed to be.
- Your analysis is reproducible. Even years later, when you've completely forgotten what you did, the whole write-up, every single number or pixel in a plot is reproducible.
- Your analysis actually works—at least in this particular instance. The code you show actually executes without error.
- Toward the end of your work, with the write-up almost done you discover an error. Months of rework to do? No! Just fix the error and rerun `knitr` and `pdflatex`. One single problem like this and you will have all the time invested in `knitr` repaid.
- This methodology provides discipline. There's nothing that will make you clean up your code like the prospect of actually revealing it to the world.

Whether we're talking about homework, a consulting report, a textbook, or a research paper. If they involve computing and statistics, this is the way to do it.