R (and RStudio) for Econometrics

Andrew P Blake

24 April 2023

Which one?

- People often ask which computer language should I use?
- But we're economists, right? So we use…. Stata? Eviews? Matlab? Gauss? Ox? C++? Fortran? Python? Julia? (Say it quietly…. Excel?)
- These are all fine, so why another? Why specifically **R**?
- It's up to you to decide, but for me *an engaged user community* is the answer. And it's free. Free is good.

R and the R GUI

- R is a **scripting language** designed to be suited to statistical analysis

- Particularly good at handling data
- Built to be a *super calculator* that does all the maths we need
- Excellent graphical and tabular output
- Multiplatform: runs on Windows, Mac, Linux and hardware as humble as the Raspberry Pi
- Free to use – simply download from https://cran.r-project.org/
- Supported by an extraordinary user community
  - https://www.r-bloggers.com/
  - https://stackoverflow.com/
  - #rstats
- Most of these resources are aimed at *data scientists*

Learning econometrics

- Typical intermediate textbooks are a good way to start, and Wooldridge (2019) leads the pack. We can use it as a template to understand how to use R
- One learning strategy is to do the example and exercises in Wooldridge (2019), which needs data
  - Data is available in an R package — `wooldridge` , see Shea (2021)
  - Code is explicitly in Heiss (2020) and available from the associated website
- Texts covering specific using R are appearing:
  - Adams (2021) for microeconometrics
  - Cunningham (2021) for causal methods
  - Huntington-Klein (2022) as an elementary text for data analysis and research design that should appeal to the budding econometrician

How to use R

- R is designed for data manipulation and analysis, often associated with data science rather than econometrics
- Data science skills are covered in Freeman and Ross (2019), and many of them are useful for us too
- Lots of helpful books to help with these sorts of skills in R – too many to mention even a fraction

Workflow

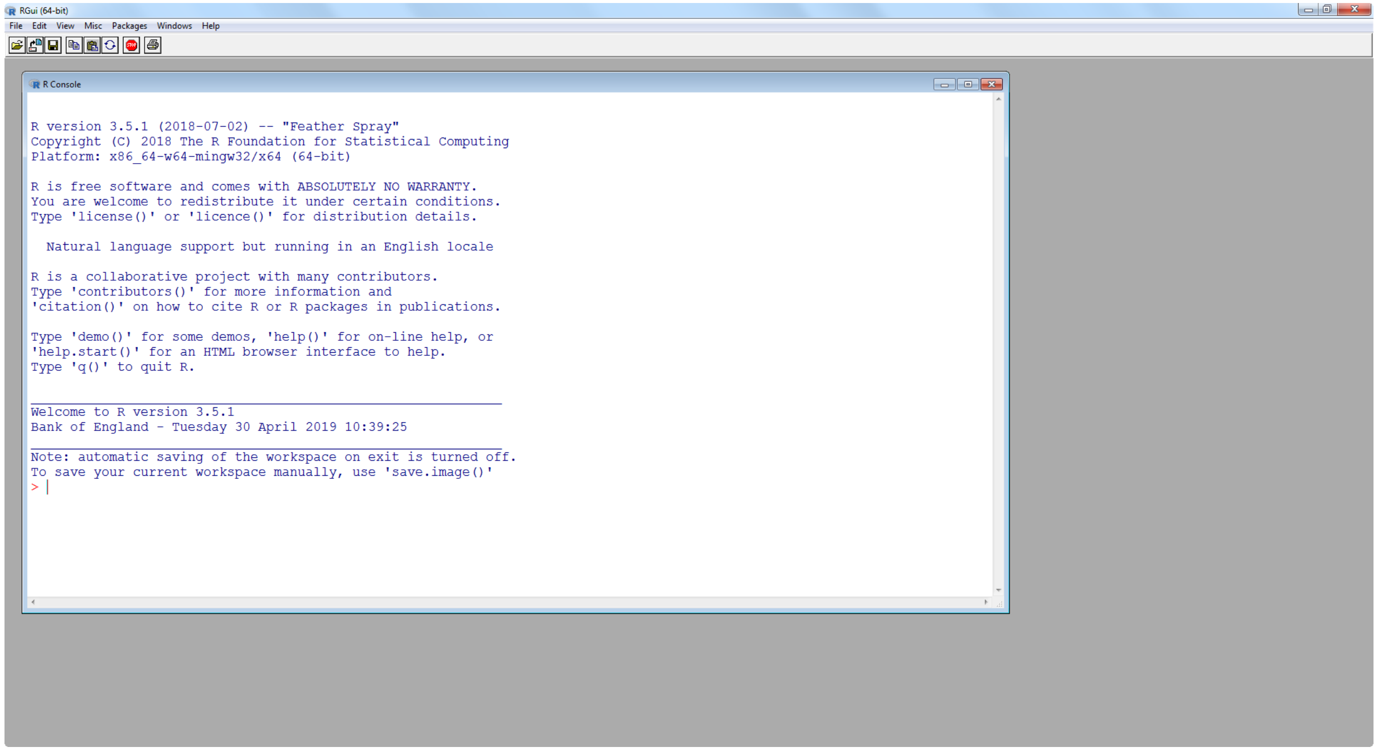
We should approach econometric analysis in this stylized way:

- Read, manipulate, clean, visualize in core/specialized packages
- Do econometric analysis either:
  - Using appropriate libraries, either part of the core packages (`lm`) or from specialist packages (say `ivreg`)
  - Or write native R code to do the analysis
- Display output, graphs etc using core/specialist R routines

The first and last of these are common to any statistical/modelling analysis, the middle bit is where the econometrician is doing something different

Basic GUI

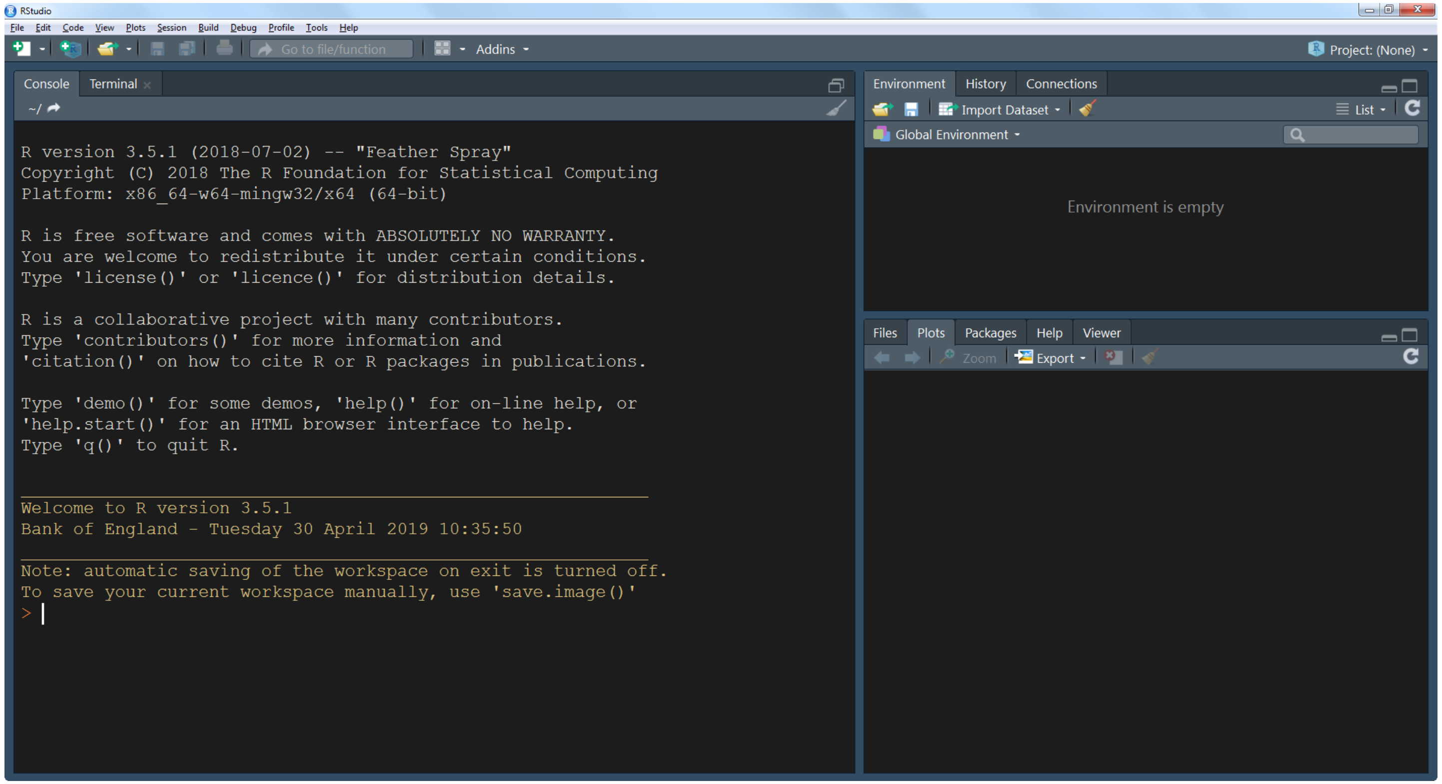
- R by itself comes with a simple but comprehensive interface:



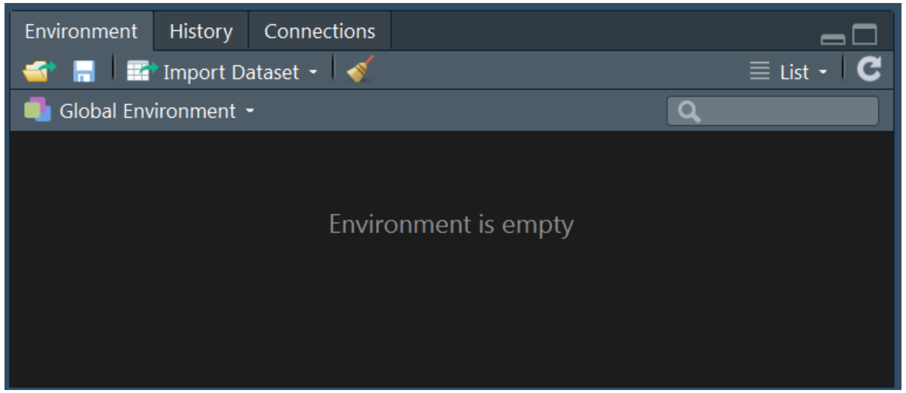
RStudio

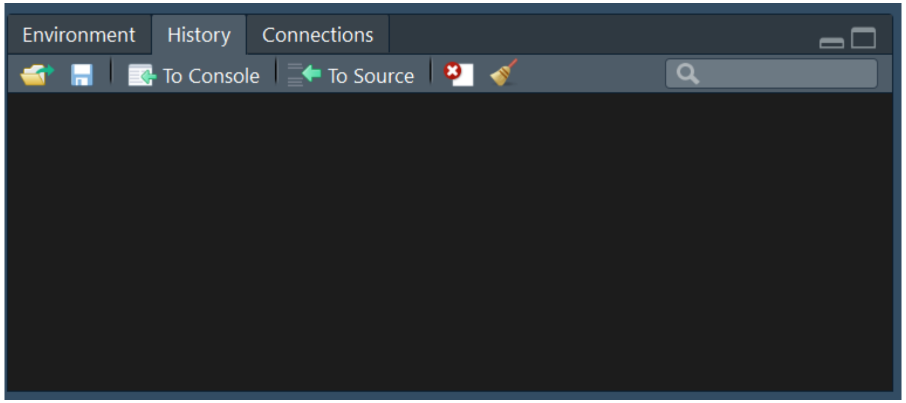
- Most people use a more comprehensive development environment such as RStudio from posit. After you download R, I would encourage you to download and install this as your GUI. It looks a lot like, say, Matlab.

- Rstudio looks something like:



- Left side as shown contains R itself
- Right side displays a lot of information:
    - Top panel, first two tabs

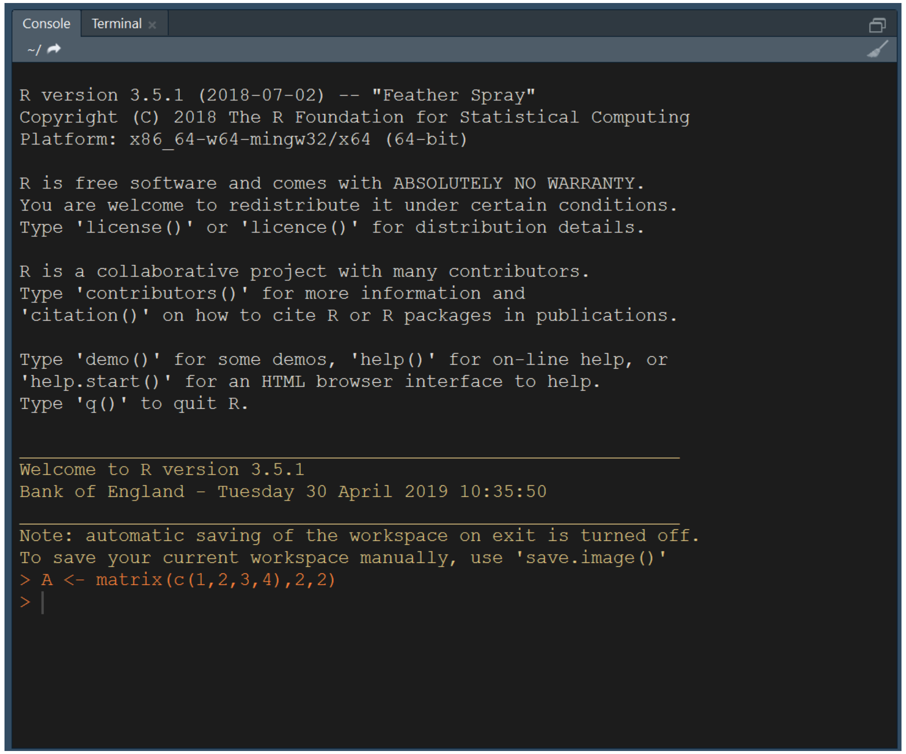




- It is very configurable, and supports lots of languages other than R. Note by default it uses a ‘black-on-white’ text colour scheme but offers any number of variations. Choose one that suits your eyes, particularly if you spend a lot of time in front of the screen!

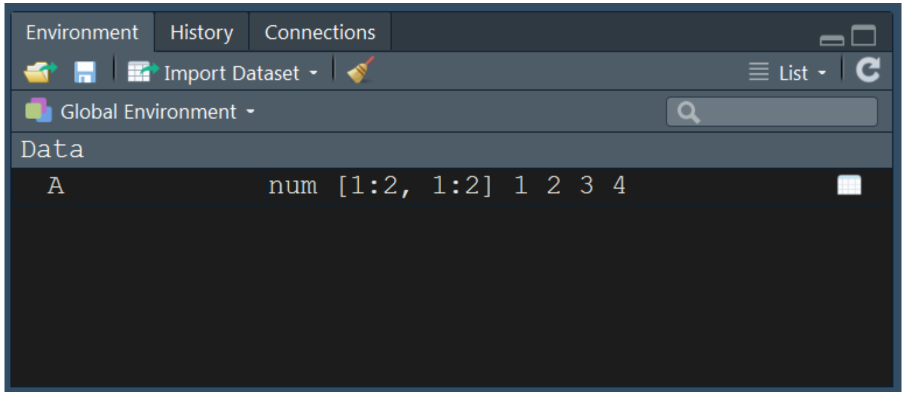
Simple first example

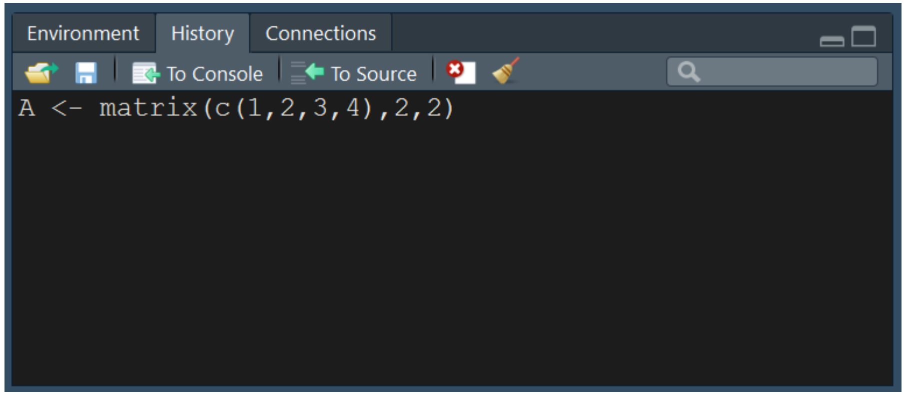
- Use R to create a matrix:




```r
A <- matrix(c(1,2,3,4), 2, 2)
A
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```





Seems easy enough: try and figure out what the function `matrix()` is doing. Change the values to all 4s, say.

Econometrics is linear algebra

That's a bit extreme, but you mostly need to do linear algebra to program up many of the estimators we use. If we wanted to *program* an estimator we need to know a bit more.

Some more linear algebra

Assume the following: A and B are real matrices of dimension $n \times n$, b and c are real $n-$vectors, X is a real $T \times k$ matrix, and S is a symmetric real matrix.

Maths commands essential to linear algebra

	Maths	R	Notes
Hadamard product	$A \odot B$	`A * B`	Element-by-element, A, B same size
Matrix/vector product	$A \times B$, $A \times b$	`A %*% B` , `A %*% b`	Normal product rule
Inner product	$X'X$	`t(X) %*% X`	Also uses transpose operator, `t()`
		`crossprod(X)`	More efficient, but less mathy
	$A'B$	`t(A) %*% B`	
		`crossprod(A,B)`	
Outer product	$A \times B'$	`tcrossprod(A,B)`	
Inverse	A^{-1}	`solve(A)`	Matrix inverse is a special case of…
Solve for d	$Ad = b \Rightarrow d = A^{-1}b$	`d <- solve(A, b)`	…linear solution!
Cholesky decomp	$S = R'R$	`R <- chol(S)`	S is a symmetric, positive definite matrix
Cholesky inverse	S^{-1}	`chol2inv(chol(S))`	Fast!

	Maths	R	Notes
Determinant	$\|A\|$	`det(A)`	
Diagonal			
of a matrix		`diag(A)`	Retrieve the elements a_{ii}, $i = 1, . ., n$
in a matrix		`A <- diag(b)`	Set the diagonal of A to b, zero elsewhere
Identity matrix	I_n	`diag(n)`	
Eigenvalues/vectors		`E <- eigen(A)`	Returns a *list*: `E$values` , `E$vectors`

Programming example: Simple linear algebra

Consider the following simultaneous system of equations:

$$\begin{aligned} x_1 + 2x_2 &= 6 \\ x_1 - 3x_2 + 2x_3 &= 0 \\ -2x_1 + 3x_3 &= 2 \end{aligned}$$

Find the values of x that solve this using R.

Hint – write the problem in matrix form

$$Ax = b$$

where

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 1 & -3 & 2 \\ -2 & 0 & 3 \end{bmatrix}, \qquad b = \begin{bmatrix} 6 \\ 0 \\ 2 \end{bmatrix}$$

and then use `solve`.

R solution

R code to create these matrices is:

```
A <- matrix(c(1,1,-2,2,-3,0,0,2,3),3,3) # Matrices are populated by column by default
b <- matrix(c(6,0,2),3,1)
```

The solution is:

```
x <- solve(A,b)
```

where `x` is:

```
##      [,1]
## [1,]    2
## [2,]    2
## [3,]    2
```

(Check it by eye!)

Precision

Take a real matrix A_{mn} with $n \leq m$ and pre-multiply by its own transpose, i.e. $B = A'A$. B is then symmetric, positive semi-definite. If $rank(A) = n$, then $rank(B) = n$ and positive definite, and its inverse exists.

```
##      [,1] [,2]
## [1,] 1.04  0.2
## [2,] 0.20  1.0
```

Let's invert B three different ways and premultiply the answers by A.

```
i1 <- solve(B)
i2 <- chol2inv(chol(B))
i3 <- qr.solve(B)

i1 %*% B
```

```
##              [,1] [,2]
## [1,] 1.000000e+00    0
## [2,] 2.775558e-17    1
```

```
i2 %*% B
```

```
##      [,1]         [,2]
## [1,]    1 5.551115e-17
```

```
## [2,]    0 1.000000e+00
```

```
i3 %*% B
```

```
##      [,1]          [,2]
## [1,]    1 -2.775558e-17
## [2,]    0  1.000000e+00
```

We see some rounding differences. This is a fundamental characteristic of numerical linear algebra.

Programming the regression problem

Let's look at the familiar regression problem for some generated data.

$$y = Xb + \epsilon$$

where $\epsilon \sim N(0, .2)$, X is a $(k+1) \times n$ matrix of regressors including a constant and b a $k+1$ vector of coefficients. Let's generate some random data of an arbitrary sized problem:

```
X <- matrix(rnorm(180, 2, 1), 60, 3)
head(X, 6) # Print first six rows
```

```
##          [,1]      [,2]     [,3]
## [1,] 2.409870 0.7925486 2.749149
## [2,] 1.710207 2.7867654 3.261341
## [3,] 3.525394 2.9972849 1.951609
```

```
## [4,] 2.966957 1.8696408 2.667315
## [5,] 1.538812 1.8504980 1.146760
## [6,] 1.640619 3.4909867 1.411701
```

```r
X <- cbind(1, X) # Add a constant
tail(X,6) # Print last six rows
```

```
##       [,1]      [,2]      [,3]      [,4]
## [55,]    1 2.2649989 2.6552562 0.8168622
## [56,]    1 0.2889892 0.2898367 2.2295481
## [57,]    1 2.5199898 2.6010970 3.7696509
## [58,]    1 1.2229275 1.0048883 2.0531373
## [59,]    1 1.0409328 2.0610489 2.4206702
## [60,]    1 2.3584212 0.9789181 1.4413930
```

Now create a dependent variable that is a linear combination of these variables plus some noise. Create the linear relationship first so we know what it is:

```r
b <- matrix(c(0.5,1,-1,.2), 4, 1)
```

and then the dependent variable:

```r
y <- X %*% b + 0.2*rnorm(60)
```

We could now do a regression - i.e. calculate

$$\hat{b} = (X'X)^{-1}X'y$$

which can be written:

```
bhat <- solve(t(X)%*%X)%*%t(X)%*%y
```

which gives

```
##              [,1]
## [1,]  0.5376433
## [2,]  0.9776018
## [3,] -0.9828396
## [4,]  0.1816978
```

But I wouldn't do it like this (we'll see why in a minute). A better way would be

```
bhat2 <- chol2inv(chol(crossprod(X)))%*%crossprod(X,y)
```

or even

```
bhat3 <- qr.solve(X,y)
```

which both evaluate to the same $\hat{b}$ values.

Testing timings

Why does it matter how you do things? It should be obvious that it might, but it turns out some fairly trivial things can make a lot of difference. We set some parameters so we can create a bigger problem.

We will use seven different methods to calculate an estimate of b. These are two variations on the three calculations below (where the brackets matter!):

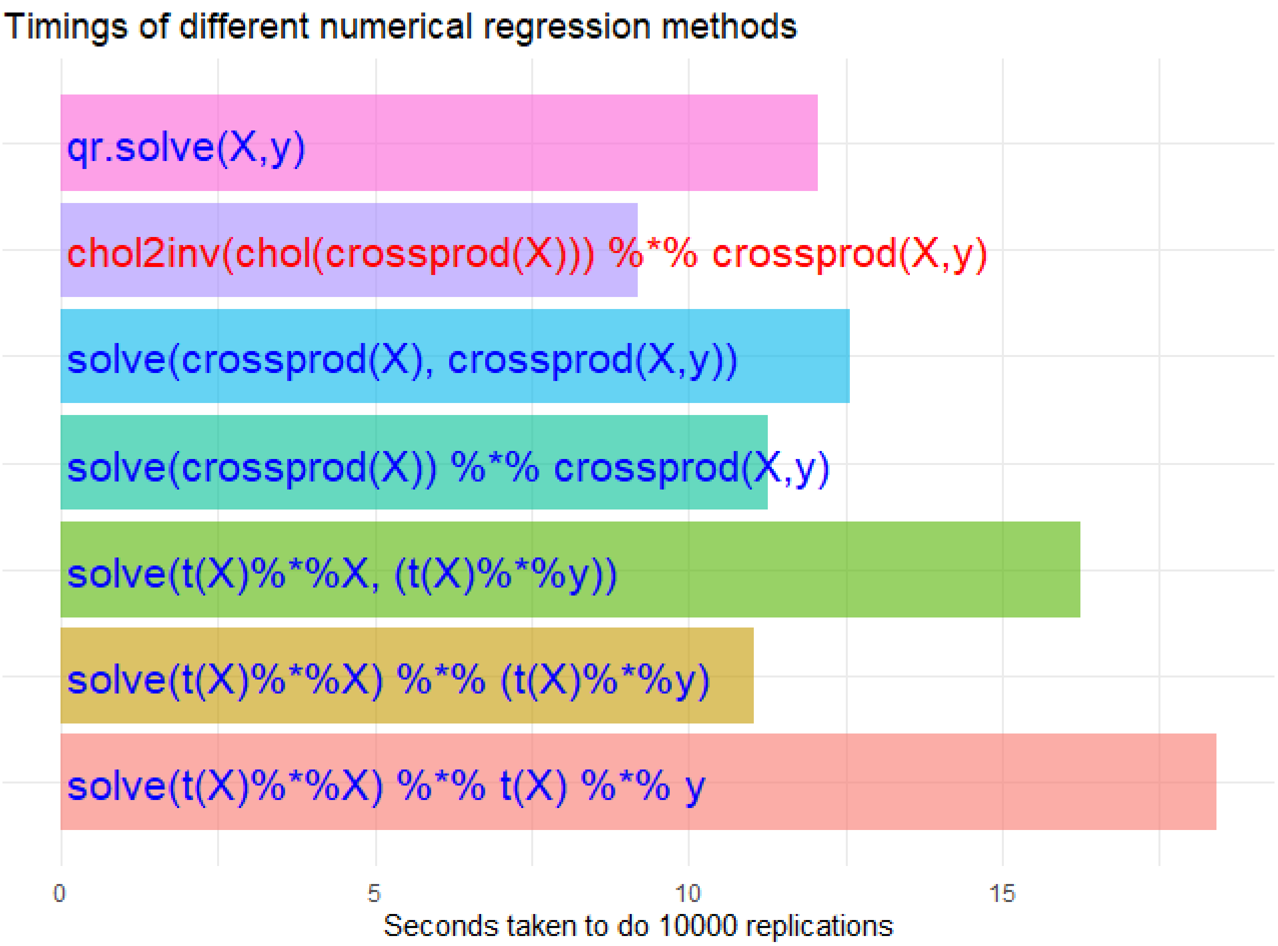
$$\hat{b}_1 = ((X'X)^{-1})X'y$$

$$\hat{b}_2 = ((X'X)^{-1})(X'y)$$

$$\hat{b}_3 = ((X'X)^{-1}(X'y))$$

where we do it either 'by hand' or using `crossprod`, plus using `qr.solve`.

The timings for these different methods are:



Using `lm` instead

- Instead of all this we could use the built in regression command `lm`

```
library(tidyverse)
data <- as_tibble(cbind(y0, X0)) %>%
  rename_all(~ c("y", "c", "X1", "X2", "X3"))
```

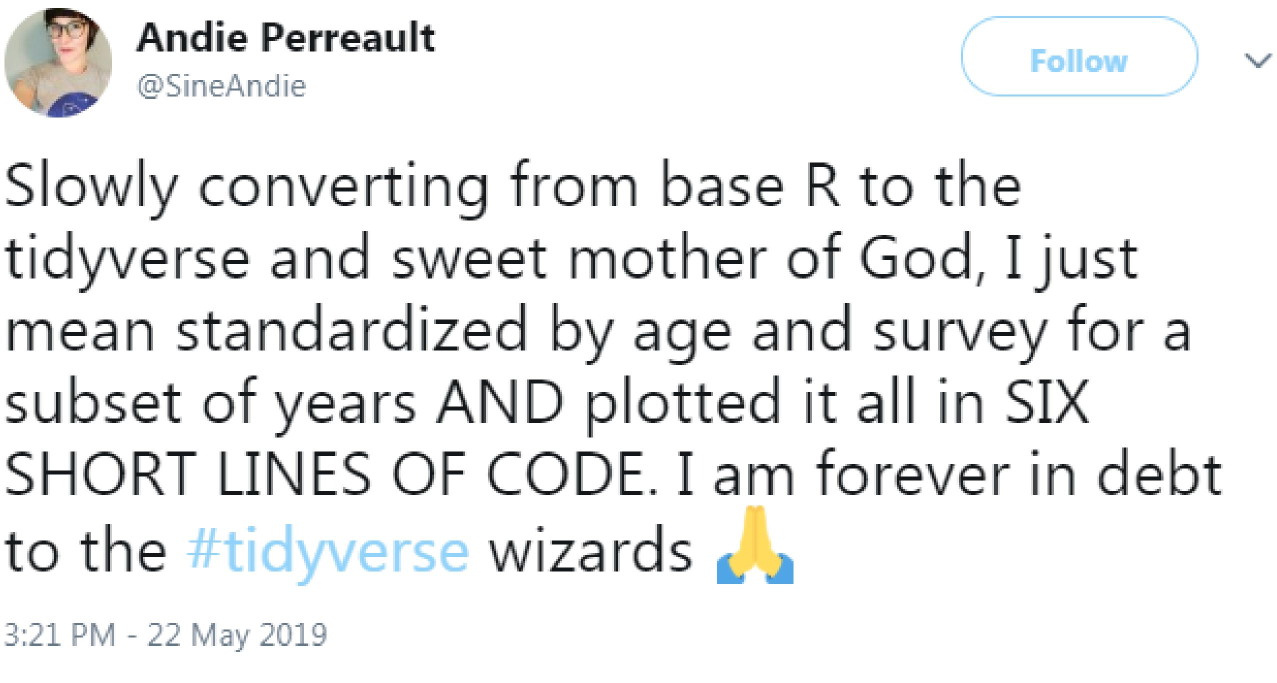
```r
reg1 <- lm(y ~ X1 + X2 + X3, data = data)
summary(reg1)
```

```
##
## Call:
## lm(formula = y ~ X1 + X2 + X3, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.35502 -0.10460 -0.00169  0.09132  0.30085
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.53764    0.07489   7.179 1.75e-09 ***
## X1           0.97760    0.01992  49.087  < 2e-16 ***
## X2          -0.98284    0.01764 -55.711  < 2e-16 ***
## X3           0.18170    0.02403   7.561 4.09e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1612 on 56 degrees of freedom
## Multiple R-squared:  0.9886, Adjusted R-squared:  0.988
## F-statistic:  1617 on 3 and 56 DF,  p-value: < 2.2e-16
```

- This does all the work for us, but if the estimator doesn’t exist, you need to do it yourself!
- Notice I load the `tidyverse`
- We need to understand every part of this

`tidyverse`

Essentially two ways to use R: with the `tidyverse` or without it. I leave it to you to decide, because although opinion is split, sometimes people have practical rather than purist responses:



Andie Perreault
@SineAndie
Follow

Slowly converting from base R to the tidyverse and sweet mother of God, I just mean standardized by age and survey for a subset of years AND plotted it all in SIX SHORT LINES OF CODE. I am forever in debt to the #tidyverse wizards 🙏

3:21 PM - 22 May 2019

This is a collection of libraries that all work together to (amongst othere things) manipulate and plot data. For a description see Wickham et al. (2019).

I use elements of the `tidyverse` throughout any code I write: in particular I often use commands from the following libraries for data wrangling and plotting:

- **dplyr**
  - **select** – retain/drop columns

    - **filter** – conditionally choose rows
    - **slice** – retain/drop rows by position
    - **mutate** – create a new variable
    - **rename** – rename an old variable
- **tidyr**
    - **pivot_longer** – make wide data long
    - **pivot_wider** – make long data wide
- **broom**
    - **tidy** – the Ronseal of the tidyverse
- **lubridate**
    - **year** – this returns the year from a date
- **magrittr**
    - **%>%** – a pipe operator that chains together commands to make manipulating data more understandable and easier to program
- **ggplot2**
    - **ggplot** – initiate a graph
    - **geom_line** – draw a line etc.

For the latter Wickham (2016) is the main reference, but learning by doing is the only way.

References

Adams, Christopher P. 2021. *Learning Microeconometrics with R*. The R Series. Oxford: CRC Press.

Cunningham, Scott. 2021. *Causal Inference: The Mixtape*. New Haven & London: Yale University Press. https://mixtape.scunning.com/.

Freeman, Michael, and Joel Ross. 2019. *Programming Skills for Data Science: Start Writing Code to Wrangle, Analyze, and Visualize Data with R*. Addison-Wesley Data and Analytics Series. New York: Pearson

Education Inc.
Heiss, Florian. 2020. *Using R for Introductory Econometrics*. 2nd ed. CreateSpace. www.URfIE.net.
Huntington-Klein, Nick. 2022. *The Effect: An Introduction to Research Design and Causality*. Routledge. https://www.theeffectbook.net/.
Shea, Justin M. 2021. *wooldridge: 115 Data Sets from "Introductory Econometrics: A Modern Approach, 7e" by Jeffrey m. Wooldridge*. https://CRAN.R-project.org/package=wooldridge.
Wickham, Hadley. 2016. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.
Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.
Wooldridge, Jeffrey. 2019. *Introductory Econometrics: A Modern Approach*. 7th ed. South-Western College Publishing.

Appendix

Some programming basics

A few not-linear-algebra commonly used commands used are summarized in the following table:

	R	Example	Notes
Assign a value	`<-`	`a <- 4`	Also legal is `a = 4`. But I hate it.
Create a list of values	`c(.)`	`v <- c(1, -2, 22)`	Defining 'on the fly'

	R	Example	Notes
Sequence	`seq(i, k, l)`	$5, 7, \ldots ,21$	Create a sequence
	`i:k`	$i, i \pm 1, \ldots ,k$	Short cut for unit in/de-crements
Loop commands	`for (var in seq) expr`	`for (i in 5:1) print(i)`	Loops. We need loops.
Draw a random number	`rnorm(k,a,b)`	`rnorm(60, 0, 5)`	Example draws 60 values ~ $N(0, 5)$
Create a matrix	`matrix(v,i,j)`	`matrix(5, 2, 2)`	Create a 2×2 matrix of 5s

Functions

Everything in R is a function (although it doesn't look like it). Defining a function is simple:

```
name_of_function <- function(function_arguments){
  # Body of function where stuff is done
}
```

Here's one that actually does something:

```
threetimesadd <- function(x,y){
  z <- 3*(x+y)
```

```r
  return(z)
}
```

and if we run it:

```r
threetimesadd(4,6)
```

```
## [1] 30
```