377 Commands

Isaac Loh

9/8/2021

R commands for 377

Getting started

To get started in RStudio, hit Ctrl+Shift+N to open up a new R script, or use File->New File->R Script.

In this class, a good way to start your scripts is with the following three commands:

```
# Clear environment
rm(list=ls())

# Set working directory
setwd("your directory here")

#Load the Wooldridge package
library(wooldridge)
```

In order, these commands clear R's working memory, set the working directory to your chosen location, and load the Wooldridge package (containing useful datasets).

Getting help

If you want to get get help with any command, you can type a ? before that command, as in the following:

```
?mean
```

```
## starting httpd help server ... done
```

You can also use the `help()` command, as in

```
help(mean)
```

Summary Statistics

In this class we will often use various summary statistics to evaluate a dataset. Now that the Wooldridge package is loaded, we can use these summary statistics to break down datasets, and their constituent variables. The summary command is a good way to succinctly express the content of a dataset. Let's try it with a dataset called cars included with R.

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
```

```
##  Max.   :25.0   Max.   :120.00
```

Evidently, the summary command tells R to display the quartiles, maximum values, minimum values, and medians for all of the variables in a dataset. If we want to view a dataset, we can use the command:

```
View(cars)
```

(case sensitive).

Given a sample for a random variable X, R can easily compute quantities like the sample mean of X ($\bar{X}$), the sample variance of X (S_X^2), and the sample standard deviation of X (S_X). **The commands in this section are only valid for sample quantities (e.g. sample means, covariances, etc).** Here is an illustration of these commands for the variable wage1$educ (note that the $ indicates that the variable name is educ, and it can be found inside the dataset named wage1 found in the Wooldridge package).

```
mean(wage1$educ)
```

```
## [1] 12.56274
```

```
var(wage1$educ)
```

```
## [1] 7.667485
```

```
sd(wage1$educ)
```

```
## [1] 2.769022
```

We can also find the covariance and correlation *between* two random variables X and Y using the following commands, evaluated on wage1$educ and wage1$wage

```
cov(wage1$wage, wage1$educ)
```

```
## [1] 4.150864
```

```
cor(wage1$wage, wage1$educ)
```

```
## [1] 0.4059033
```

Note that R calculates that, in the wage1 dataset, the sample covariance between wage and education (4.150864) is positive, which makes sense. The sample covariance determines the sign of the correlation between two variables by the formula

$$\rho_{XY} = \frac{S_{XY}}{S_X S_Y},$$

so we know that the sample correlation between X and Y must also be positive (it's 0.4059033).

Manipulating data

R can help you manipulate data and compute quantities based on the data. Suppose we have a random sample of die rolls X given by $\{X_i : i = 1, \ldots, 5\} = \{1, 2, 4, 2, 6\}$. This tells us that we have a sample size of $n = 5$, and that the observations we received were 1,2,4,2,6. We can enter this into R as a *vector* (ordered list of numbers):

```
c(1,2,4,2,6)
```

```
## [1] 1 2 4 2 6
```

The `c` command combines its arguments to form a vector, which is just an ordered list of numbers. You cannot enter a list of numbers into R in parentheses without the `c`. Note that if we want to enter a string of successive numbers, we can use a colon with the start and end numbers. For example:

```
c(1:5)
```

```
## [1] 1 2 3 4 5
```

```
c(6:2)
```

```
## [1] 6 5 4 3 2
```

We can define a vector `X` to equal our data (which will simplify handling the data later on) using the syntax:

```
X = c(1,2,4,2,6)
```

When you present R with a command of the form `a = b`, R will create a new object `a` equal to `b`, or redefine `a` to be equal to `b` if `a` already exists.

Now that we have `X` defined, we can use it interchangeably with what we set it equal to. For instance,

```
mean(c(1,2,4,2,6))
```

```
## [1] 3
```

which calculates the sample mean of our die rolls, will give the same result as

```
mean(X)
```

```
## [1] 3
```

which is much easier to type. We can also perform basic arithmetic with vectors:

```
X
```

```
## [1] 1 2 4 2 6
```

```
X + 1
```

```
## [1] 2 3 5 3 7
```

```
X * 2
```

```
## [1]  2  4  8  4 12
```

```
X ** 2
```

```
## [1]  1  4 16  4 36
```

```
exp(X)
```

```
## [1]   2.718282   7.389056  54.598150   7.389056 403.428793
```

Given a new vector `Y` of the same dimension as `X`, we can add `X + Y`, subtract `X - Y`, multiply `X * Y`, and divide `X / Y`:

```
Y = c(2,4,3,1,5)
X + Y
```

```
## [1]  3  6  7  3 11
```

```
X - Y
```

```
## [1] -1 -2  1  1  1
```

```
X * Y
```

```
## [1]  2  8 12  2 30
```

```
X / Y
```

```
## [1] 0.500000 0.500000 1.333333 2.000000 1.200000
```

Manipulating data within datasets

R allows us to manipulate data within datasets just as we can with vectors. We can create new datasets given existing data using an `=`:

```
mydata = wage1
```

To view a list of the random variables in a dataset, use the `ls()` command:

```
ls(mydata)
```

```
##  [1] "clerocc"  "construc" "educ"     "exper"    "expersq"  "female"
##  [7] "lwage"    "married"  "ndurman"  "nonwhite" "northcen" "numdep"
## [13] "profocc"  "profserv" "services" "servocc"  "smsa"     "south"
## [19] "tenure"   "tenursq"  "trade"    "trcommpu" "wage"     "west"
```

To count the number of variables in a dataset, use the `ncol()` command. To count the number of observations in a dataset, use the `nrow` command:

```
ncol(mydata)
```

```
## [1] 24
```

```
nrow(mydata)
```

```
## [1] 526
```

This tells us that, in the dataset `mydata`, there are `rncol(mydata)‘ variables and $n = $ 526 observations.

To create a new variable inside `mydata`, use the `$` notation in conjunction with an `=`:

```
mydata$new_variable = mydata$wage
```

After running this command, we will have a new variable (here, titled `new_variable`) created inside of `mydata`:

```
ls(mydata)
```

```
##  [1] "clerocc"      "construc"     "educ"         "exper"        "expersq"
##  [6] "female"       "lwage"        "married"      "ndurman"      "new_variable"
## [11] "nonwhite"     "northcen"     "numdep"       "profocc"      "profserv"
## [16] "services"     "servocc"      "smsa"         "south"        "tenure"
## [21] "tenursq"      "trade"        "trcommpu"     "wage"         "west"
```

We can perform arithmetic operations on variables, or between variables, just as we did with vectors:

```
mydata$wage + 1
mydata$wage * 2
mydata$wage**2
mydata$wage + mydata$educ
```

and we can define new variables using these operations:

```
mydata$wagesq = mydata$wage**2
```

Using R to compute population parameters

R's usefulness as a calculator also allows us to compute population level parameters using joint distribution tables. Suppose that we have the joint distribution specified in Table 1 for random variables X and Y (note that, because we have a joint distribution given to us, we are dealing with a population level question):

Table 1:

$\mathrm{P}(X = x, Y = y)$	1/4	1/2	1/12	1/12	1/12
x	1	2	3	4	5
y	6	5	4	3	2

Table 2:

$\mathrm{P}(X = x, Y = y)$	1/4	1/2	1/12	1/12	1/12
x	1	2	3	4	5
y	6	5	4	3	2
xy	6	10	12	12	10

The first step to handle computing population level questions would be to input the probabilities as a vector, and then the set of values of x and y as their own vectors, *preserving the order in which they appear in the joint distribution table*.

```
probvec = c(1/4, 1/2, 1/12, 1/12, 1/12)
xvec = c(1, 2, 3, 4, 5)
yvec = c(6, 5, 4, 3, 2)
```

The formula for e.g. the expected value $\mathrm{E}[X]$ of X is given by

$$\sum_x x \cdot \mathrm{P}(X = x). \tag{1}$$

Using our definitions, we can get a new vector containing as entries the products $x \cdot \mathrm{P}(X = x)$ using the `*` operator as in the following command:

```
xvec * probvec
```

```
## [1] 0.2500000 1.0000000 0.2500000 0.3333333 0.4166667
```

According to formula (1), the only remaining step to find $\mathrm{E}[X]$ is to take the sum of the vector above. This can be done using the `sum()` command, which sums up the values of a list of numbers (be it a vector or a column of a dataset):

```
sum(xvec * probvec)
```

```
## [1] 2.25
```

Hence, in this example, $\mathrm{E}[X] = 2.25$

Likewise, to get the expected value $\mathrm{E}[XY]$ of the random variable XY, we would create a new row in our probability distribution table containing products of the form xy, as in Table 2.

This new row (vector) can be obtained in R using the following command:

```
xvec * yvec
```

```
## [1]  6 10 12 12 10
```

The last step to evaluate for $\mathrm{E}[XY]$ would be to multiply the elements in this vector by the vector of probabilities, and sum the entries. We can do this in the following way:

```
sum((xvec * yvec) * probvec)
```

```
## [1] 9.333333
```

Hence, $\mathrm{E}[XY] = 9.3333333$. The commands we could use to compute several quantities we care about in this class are summarized below:

```
EX = sum(xvec * probvec)
EY = sum(yvec * probvec)
EXsq = sum(xvec**2 * probvec)
EYsq = sum(yvec**2 * probvec)
EXY = sum(xvec * yvec * probvec)
```

These commands calculate $\mathrm{E}[X]$, $\mathrm{E}[Y]$, $\mathrm{E}\left[X^2\right]$, $\mathrm{E}\left[Y^2\right]$, and $\mathrm{E}[XY]$, respectively, using the information contained in Table 1. The values are summarized below:

Population parameter	Computed value
`EX`	2.25
`EY`	4.75
`EXsq`	6.4166667
`EYsq`	23.9166667
`EXY`	9.3333333

Using the parameters that we just calculated, we can now determine the variances of X and Y, as well as their covariance and correlation. Recall that:

$$\begin{aligned}
\mathrm{Var}(X) &= \mathrm{E}\left[X^2\right] - \mathrm{E}[X]^2 \\
\mathrm{sd}(X) &= \sqrt{\mathrm{Var}(X)} \\
\mathrm{Cov}(X,Y) &= \mathrm{E}[XY] - \mathrm{E}[X] \cdot \mathrm{E}[Y] \\
\mathrm{Cor}(X,Y) &= \frac{\mathrm{Cov}(X,Y)}{\mathrm{sd}(X)\,\mathrm{sd}(Y)}
\end{aligned}$$

Using these formulas, we can use the following commands to get $\mathrm{Var}(X)$, $\mathrm{sd}(X)$, $\mathrm{Var}(Y)$, $\mathrm{sd}(Y)$, $\mathrm{Cov}(X,Y)$, and $\mathrm{Cor}(X,Y)$ respectively:

```
varX = EXsq - EX**2
sdX = sqrt(varX)
varY = EYsq - EY**2
sdY = sqrt(varY)
covXY = EXY - EX * EY
corXY = covXY / (sdX * sdY)
```

Population parameter	Computed value
`varX`	1.3541667
`sdX`	1.1636867
`varY`	1.3541667
`sdY`	1.1636867
`covXY`	-1.3541667
`corXY`	-1

Note that the correlation is always between -1 and 1.

Linear regression

Simple linear regression Given two lists of numbers of equal length, be they vectors or variables in a dataset, R can easily regress one (the dependent variable) on the other. The command to do this is `lm`:

```
lm(wage1$wage ~ wage1$educ)
```

```
##
## Call:
## lm(formula = wage1$wage ~ wage1$educ)
##
## Coefficients:
## (Intercept)   wage1$educ
##     -0.9049       0.5414
```

Instead of using `$` notation every time a variable is used inside the `lm()` command, one can just use the option to specify the dataset name at the end of the command:

```
lm(wage ~ educ, data = wage1)
```

```
##
## Call:
## lm(formula = wage ~ educ, data = wage1)
##
## Coefficients:
## (Intercept)         educ
##     -0.9049       0.5414
```

When we use the `lm()` command for a simple linear regression, R prints out the estimated intercept coefficient $\hat{\beta}_0$ (in this case, -0.9048516) and the estimated slope coefficient $\hat{\beta}_1$ (in this case, 0.5413593). It also computes many other quantities related to the regression which we can access by saving the regression output. For example, if we wanted to save the regression as `reg`, we would type

```
reg = lm(wage ~ educ, data = wage1)
```

We can now access features of our regression output using the `$` syntax. For instance:

```
reg$coefficients
```

```
## (Intercept)        educ
##  -0.9048516   0.5413593
```

returns $\hat{\beta}_0$ and $\hat{\beta}_1$.

```
reg$coefficients[1]
```

```
## (Intercept)
##  -0.9048516
```

returns the first element of the coefficient vector, which is $\hat{\beta}_0$.

```
reg$coefficients[2]
```

```
##      educ
## 0.5413593
```

returns the second element of the coefficient vector, which is $\hat{\beta}_1$.

```
reg$residuals
```

returns a list of the *residuals* from the regression, so the sum of squared residuals can be found using the command

```
sum(reg$residuals**2)
```

```
## [1] 5980.682
```

This command tells us that, for this regression, SSR = 5980.6822547. R also retains the *predicted*, or *fitted*, values for Y, denoted $\hat{Y}$. These are given by the formula

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i,$$

and can be accessed with the command

```
reg$fitted.values
```

We can also conveniently get a more detailed breakdown of the regression results by using the `summary()` command:

```
summary(reg)
```

```
##
## Call:
## lm(formula = wage ~ educ, data = wage1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.3396 -2.1501 -0.9674  1.1921 16.6085
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.90485    0.68497  -1.321    0.187
## educ         0.54136    0.05325  10.167   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.378 on 524 degrees of freedom
## Multiple R-squared:  0.1648, Adjusted R-squared:  0.1632
## F-statistic: 103.4 on 1 and 524 DF,  p-value: < 2.2e-16
```

The summary of our regression will contain more useful information. For instance, we can get the R^2 of the regression with the command

```
summary(reg)$r.squared
```

```
## [1] 0.1647575
```