

Week 2: R Tutorial

ResEcon 703: Topics in Advanced Econometrics

Matt Woerman
University of Massachusetts Amherst

Agenda

Last week

- Structural estimation

This week's topics

- R resources
- Objects in R
- Functions and packages in R
- Math and statistics in R
- Data in R
- R examples

This week's "reading"

- R `swirl` interactive tutorials

R Resources

Hat Tips



This lecture is inspired heavily by notes and slides created by

- Fiona Burlig, University of Chicago
- Grant McDermott, University of Oregon
- Ed Rubin, University of Oregon

Many thanks to them for generously making their course materials available online for all!

Installing R

Installing R is *usually* straightforward

-  Download (cran.r-project.org) and install R
-  Download (www.rstudio.com/products/rstudio/download) and install RStudio Desktop (Open Source License)

What is the difference between R and RStudio?



R is like a car's engine. It is the program that powers your data analysis.



RStudio is like a car's dashboard. It is the program you interact with to harness the power of your “engine.”

R swirl Interactive Tutorials

swirl is an R package that interactively teaches you how to use R

- Information available here: swirlstats.com

```
## Install swirl package
install.packages('swirl')
## Load swirl package
library(swirl)
## Install swirl tutorials
install_course('R Programming')
install_course('Getting and Cleaning Data')
install_course('Advanced R Programming')
## Start swirl tutorials
swirl()
```

These three swirl tutorials (R Programming, Getting and Cleaning Data, and Advanced R Programming) introduce the main R concepts we will use in this course

More R Resources

These links provide a variety of perspectives and topics related to using R for statistical analysis, all of which may be useful as you learn to use R for structural estimation in this course

- DataCamp's Introduction to R
- R for Data Science book
- Advanced R book
- Ed Rubin's Econometrics lab slides
- Ed Rubin's Econometrics section notes
- Fiona Burlig's Econometrics section notes (warning: puns ahead)
- Grant McDermott's Data Science for Economists lecture slides

Some Complements to R

\LaTeX and knitr

- \LaTeX (www.latex-project.org): Typesetting system with great functionality for technical and scientific documents
- knitr (yihui.name/knitr): R package that integrates R code and output into \LaTeX documents (or HTML, Markdown, etc.)

Git, GitHub, and SmartGit

- Git (git-scm.com): Version control system
- GitHub (github.com): Hosting platform for Git
 - ▶ Some alternatives exist: BitBucket, SourceForge, GitLab
- SmartGit (www.syntevo.com/smartgit): GUI client for Git
 - ▶ Many alternatives exist: GitHub Desktop, GitKraken, SourceTree

Objects in R

Object Basics

Everything is an object, and every object has a name and value

```
## Assign a value of 1 to an object called a  
a <- 1  
## Assign a value of 2 to an object called b  
b <- 2  
## You use these objects in operations and functions  
a + b  
## [1] 3  
  
## Assign object c to have a value equal to a + b  
c <- a + b  
c  
## [1] 3
```

Classes, Types, and Structures

Every object has a type

- Numeric: 1, 0.5, 2/3, pi
- Character: "Hello", "cruel world", "Metrics is fun!"
- Logical: TRUE, FALSE, T, F

Every object has a structure

- Vector
- Matrix
- List
- Data frame

`class()`, `typeof()`, `str()` give information about an object

Vectors

A vector is a collection of elements of the same type

- `c()` combines elements into a vector
- `seq()` and `:` create sequential vectors of numeric elements

```
## Create a numeric vector
c(1, 1, 2, 3, 5, 8, 13)
## [1] 1 1 2 3 5 8 13

## Create a sequential vector
0:9
## [1] 0 1 2 3 4 5 6 7 8 9

## Create a character vector
c('Hello', 'world')
## [1] "Hello" "world"
```

If you combine elements of different types, R will convert some

```
## Create a vector with numeric, character, and logical elements
c(1, 'Hello', 3, 'world', TRUE)
## [1] "1"      "Hello" "3"      "world" "TRUE"
```

Matrices

A matrix is a collection of elements of the same type arranged in two dimensions

`matrix()` arranges a vector of data into a matrix

- `data`: Vector of data to create matrix
- `nrow` or `ncol`: Number of rows or columns in the matrix
- `byrow`: Logical indicating how to arrange data

```
## Create a 2 (rows) x 5 (columns) matrix of 1:10 arranged by row
matrix(data = 1:10, nrow = 2, byrow = TRUE)
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
```

Lists

A list is a collection of elements that can have different types and different structures

`list()` combines elements into a list

```
## Create a list with a numeric vector, matrix, and character vector
list(c(2, 4, 6, 8), matrix(1:4, 2), c('a', 'b', 'c'))
## [[1]]
## [1] 2 4 6 8
##
## [[2]]
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## [[3]]
## [1] "a" "b" "c"
```

Data Frames

A data frame is a structured table of data arranged in two dimensions

- Each column is a “variable” and each row is an “observation”
- Technically, a data frame is a list of named vectors of the same length
 - ▶ Each vector is a “variable”
 - ▶ The length of each vector equals the number of “observations”

`data.frame()` combines vectors into a data frame

```
## Create a data frame with 4 variables and 3 observations
data.frame(x = 0:2, y = c(2, 4, 8), z = c(1, 5, 7), w = c('a', 'b', 'c'))
##   x y z w
## 1 0 2 1 a
## 2 1 4 5 b
## 3 2 8 7 c
```

Functions and Packages in R

Functions

A function in R

- 1 Takes some inputs
- 2 Performs some internal tasks
- 3 Returns some output

We have already seen some examples of functions

- `matrix()`
 - 1 Takes a vector of data, information about the size of the matrix, and information about the arrangement of the matrix
 - 2 Arranges the data in the way specified by the other inputs
 - 3 Returns a matrix object

Use `?` (e.g., `?matrix`) to get the help file for a function

Function Inputs

Many functions have default inputs so you do not have to specify all the arguments

- These defaults are shown when you look at the function help file

Use `?matrix` to see the set of default inputs for the `matrix()` function

```
## Matrix function default inputs  
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
```

So the default inputs would create a 1×1 matrix of NA

```
## Create matrix with default inputs  
matrix()  
##      [,1]  
## [1,]  NA
```

Inputs can also be highly flexible

- `c()` allows for any number of arguments (as long as you have the memory to create a vector of the specified length)

User-Defined Functions

R makes it easy to define your own functions

Why create your own functions?

- You are performing the same task more than once
- You want to make it easier to parallelize your code
- You want to make your code more readable

How to create your own functions using `function(){}`

- 1 Specify the inputs in the `()`
- 2 Write the code for the function tasks in the `{}`
- 3 Specify the output using `return()` in the `{}`

Function Example

Make a function that calculates the mean sum of squares of three numbers

```
## Define a function that calculates the MSS from three inputs
mean_sum_squares <- function(num1, num2, num3){
  ## Calculate the mean sum of squares
  mss <- (num1^2 + num2^2 + num3^2) / 3
  ## Return the answer
  return(mss)
}
```

Try it out

```
## Calculate the mean sum of squares of 1, 2, and 3
mean_sum_squares(1, 2, 3)
## [1] 4.666667
```

What if we want a default argument?

```
## Make 3 the default input for the third argument
mean_sum_squares <- function(num1, num2, num3 = 3)
```

What if we want a flexible number of inputs?

- That is a little more complicated and context-specific...

Packages

A package is a bundle of code, documentation, and data that has been created and distributed by another R user

- More than 16,000 packages are available on CRAN, the official repository of R packages

What is so great about packages?

- Packages greatly increase the functionality available to you through “canned” routines
- Packages are open source
 - ▶ A package can be created by anyone, even you!
 - ▶ You can see the source code in any package
- Some packages have vignettes that provide detailed examples for using the package’s functionality

Any problems to be aware of?

- A package can be created by anyone, so *caveat utilitor* (user beware)

Using Packages

First download a package from CRAN using `install.packages()`

```
## Install a few packages we will use in this course  
install.packages(c('tidyverse', 'mlogit', 'gmm'))
```

Then load the package into your R session using `library()`

```
## Load those packages  
library(tidyverse)  
library(mlogit)  
library(gmm)
```

Update packages occasionally using `update.packages()`

Recommended Packages

Packages we will use in this course

- `tidyverse`
 - ▶ Collection of packages that improve data analysis and visualization
- `mlogit`
 - ▶ Estimating multinomial logit models
- `gmm`
 - ▶ Generalized method of moments estimation

Other good packages

- `glue`
 - ▶ Character functions
- `lubridate`
 - ▶ Date and time functions
- `lfe`
 - ▶ Fixed effects models
- `furrr`
 - ▶ Parallelization

Math and Statistics in R

Math Operations

Addition

$a + b$

[1] 3

Subtraction

$a - b$

[1] -1

Multiplication

$a * b$

[1] 2

Division

a / b

[1] 0.5

Exponents

a^b

[1] 1

Math Functions

```
## Absolute value
```

```
abs(a - b)
```

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

```
## Exponential
```

```
exp(a)
```

```
## [1] 2.718282
```

```
## Square root
```

```
sqrt(b)
```

```
## [1] 1.414214
```

```
## Natural log
```

```
log(b)
```

```
## [1] 0.6931472
```

```
## Log base 10
```

```
log(b, base = 10)
```

```
## [1] 0.30103
```

Statistics Functions

```
## Create a vector 0 to 4
v <- 0:4
v
## [1] 0 1 2 3 4

## Mean
mean(v)
## [1] 2

## Median
median(v)
## [1] 2

## Standard deviation
sd(v)
## [1] 1.581139
```

Sampling Functions

```
## Set the seed for randomization
set.seed(703)
## Draw from a random normal  $N(3, 2)$ 
rnorm(n = 5, mean = 3, sd = sqrt(2))
## [1] 1.142567 4.223916 1.236003 3.846436 1.268874

## Draw with replacement from v
sample(v, size = 10, replace = TRUE)
## [1] 1 0 1 3 1 0 4 1 4 0

# CDF of a standard normal at  $z = 1.96$ 
pnorm(q = 1.96, mean = 0, sd = 1)
## [1] 0.9750021
```

Vectorization

Many operations and functions are applied to each element of a vector

```
## Addition with each element
```

```
v + a
```

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

```
## Multiplication with each element
```

```
v * b
```

```
## [1] 0 2 4 6 8
```

```
## Exponential of each element
```

```
exp(v)
```

```
## [1] 1.000000 2.718282 7.389056 20.085537 54.598150
```

```
## Natural log of each element
```

```
log(v)
```

```
## [1] -Inf 0.000000 0.6931472 1.0986123 1.3862944
```

Vector Math

You can also operate on vectors elementwise

```
## Elementwise addition
```

```
v + 1:5
```

```
## [1] 1 3 5 7 9
```

```
## Elementwise multiplication
```

```
v * 1:5
```

```
## [1] 0 2 6 12 20
```

But weird things can happen if the vectors are different lengths

```
## Elementwise addition with different lengths
```

```
v + 1:4
```

```
## Warning in v + 1:4: longer object length is not a multiple of  
shorter object length
```

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

Indexing Vectors

Access elements within a vector using `[]`

```
## Access the second element of v
```

```
v[2]
```

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

```
## Access the second and fourth elements of v
```

```
v[c(2, 4)]
```

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

```
## Access all but the first element of v
```

```
v[-1]
```

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

```
## Replace the first element of v with 5
```

```
v[1] <- 5
```

```
v
```

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

Matrices as Vectors

Matrices (usually) work like vectors

```
## Create a matrix
m <- matrix(1:4, nrow = 2)
m
##           [,1] [,2]
## [1,]        1    3
## [2,]        2    4

## Mean
mean(m)
## [1] 2.5

## Natural log of each element
log(m)
##           [,1]      [,2]
## [1,] 0.0000000 1.098612
## [2,] 0.6931472 1.386294
```


Matrix Addition

Matrix addition and subtraction is performed elementwise

```
## Create a second matrix
n <- matrix(c(2, 4, 6, 8), nrow = 2)
n
##      [,1] [,2]
## [1,]    2    6
## [2,]    4    8

## Matrix addition
m + n
##      [,1] [,2]
## [1,]    3    9
## [2,]    6   12
```

Matrix Multiplication

Using `*` to multiply matrices performs elementwise multiplication

```
## Elementwise matrix multiplication
m * n
##      [,1] [,2]
## [1,]    2  18
## [2,]    8  32
```

You must use `%*%` to get the matrix product

```
## Matrix product
m %*% n
##      [,1] [,2]
## [1,]   14  30
## [2,]   20  44
```

Matrix Functions

R has many other functions for use with matrices

```
## Transpose
```

```
t(m)
```

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

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

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

```
## Inverse
```

```
solve(m)
```

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

```
## [1,]   -2  1.5
```

```
## [2,]    1 -0.5
```

Indexing Matrices

Access elements within a matrix using []

```
## Access the element in the second row and first column of m  
m[2, 1]  
## [1] 2  
  
## Access the first row of m  
m[1, ]  
## [1] 1 3  
  
## Access the second column of m  
m[, 2]  
## [1] 3 4
```

Data in R

Example Data Frame

You will mostly interact with datasets in the form of data frames

- R includes several example data frames

```
## Show an example data frame, mtcars
```

```
head(mtcars)
```

##	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
## Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
## Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
## Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
## Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
## Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
## Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

Indexing Data Frames

Access elements within a data frame using []

```
## Access the third observation of mtcars
```

```
mtcars[3, ]
```

```
##           mpg cyl  disp  hp  drat   wt  qsec vs am gear carb
## Datsun  710 22.8   4  108  93 3.85 2.32 18.61  1  1    4    1
```

```
## Access the second variable of mtcars
```

```
mtcars[, 2]
```

```
## [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
```

Access a variable of a data frame using \$

```
## Access the cyl variable of mtcars
```

```
mtcars$cyl
```

```
## [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
```

Adding New Variables

You may want to add new variables to a data frame

```
## Add an id variable to mtcars
mtcars$id <- 1:nrow(mtcars)
## Add a variable that is the power-to-weight ratio (hp / wt)
mtcars$ptw <- mtcars$hp / mtcars$wt
head(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
## Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
## Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
## Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
## Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
## Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
## Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1


```
##
##          id      ptw
## Mazda RX4      1 41.98473
## Mazda RX4 Wag  2 38.26087
## Datsun 710      3 40.08621
## Hornet 4 Drive  4 34.21462
## Hornet Sportabout 5 50.87209
## Valiant        6 30.34682
```

But that can get a little clunky. Is there a better way?

dplyr

dplyr is a package that greatly improves data manipulation in R

- Part of the tidyverse so it is already installed and loaded from earlier code

dplyr is a “grammar of data manipulation”

- Data compose the subjects of your analysis
- dplyr provides the verbs
 - ▶ `mutate()`: Adds new variables
 - ▶ `select()`: Picks variables
 - ▶ `filter()`: Picks observations
 - ▶ `arrange()`: Changes the order of observations
 - ▶ `summarize()` or `summarise()`: Summarizes multiple observations

Adding New Variables with dplyr

```
mutate(.data, ...)
```

- .data: Existing data frame
- ...: Names and values of new variables

```
## Add id and power-to-weight ratio variables
```

```
mtcars <- mutate(mtcars, id = 1:n(), ptw = hp / wt)
```

```
head(mtcars)
```

##	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
## Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
## Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
## Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
## Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
## Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
## Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

##	id	ptw
## Mazda RX4	1	41.98473
## Mazda RX4 Wag	2	38.26087
## Datsun 710	3	40.08621
## Hornet 4 Drive	4	34.21462
## Hornet Sportabout	5	50.87209
## Valiant	6	30.34682

Tibbles

tidyverse also introduces a new kind of data frame, the tibble

- Actually, `tibble` is the name of the package that has the code to create and manipulate objects of class `tbl_df`
- But it is easier to say “tibble,” so that is what users call both the package and the object
- I will probably use “tibble” and “data frame” interchangeably to mean “tibble”

Why are tibbles better than data frames?

- Data frames sometimes exhibit weird behaviors related to naming variables or trying to convert variable types
- Tibbles are smarter about how much data they show you when you call them
 - ▶ You do not have to use `head()` to suppress output

Example Tibble

dplyr comes with several examples tibbles

```
## Show an example tibble, starwars
```

```
starwars
```

```
## # A tibble: 87 x 14
```

```
##   name    height    mass hair_color skin_color eye_color birth_year sex
##   <chr>   <int>   <dbl> <chr>      <chr>      <chr>      <dbl> <chr>
## 1 Luke~    172     77 blond      fair        blue        19    male
## 2 C-3PO    167     75 <NA>      gold        yellow     112    none
## 3 R2-D2     96     32 <NA>      white, bl~  red        33    none
## 4 Dart~    202    136 none       white       yellow     41.9  male
## 5 Leia~    150     49 brown      light       brown       19    fema~
## 6 Owen~    178    120 brown, gr~ light       blue       52    male
## 7 Beru~    165     75 brown      light       blue       47    fema~
## 8 R5-D4     97     32 <NA>      white, red  red        NA    none
## 9 Bigg~    183     84 black      light       brown       24    male
## 10 Obi~-    182     77 auburn, w~ fair        blue-gray   57    male
## # ... with 77 more rows, and 6 more variables: gender <chr>,
## #   homeworld <chr>, species <chr>, films <list>, vehicles <list>,
## #   starships <list>
```

Let's play around with the dplyr verbs on this tibble

select() Example

```
## Select name, homeworld, and species in starwars
```

```
select(starwars, name, homeworld, species)
```

```
## # A tibble: 87 x 3
```

```
##   name                homeworld species
```

```
##   <chr>                <chr>      <chr>
```

```
## 1 Luke Skywalker      Tatooine  Human
```

```
## 2 C-3P0               Tatooine  Droid
```

```
## 3 R2-D2               Naboo     Droid
```

```
## 4 Darth Vader         Tatooine  Human
```

```
## 5 Leia Organa         Alderaan Human
```

```
## 6 Owen Lars           Tatooine  Human
```

```
## 7 Beru Whitesun lars  Tatooine  Human
```

```
## 8 R5-D4               Tatooine  Droid
```

```
## 9 Biggs Darklighter  Tatooine  Human
```

```
## 10 Obi-Wan Kenobi     Stewjon   Human
```

```
## # ... with 77 more rows
```

filter() Example

```
## Filter to show only droids in starwars
filter(starwars, species == 'Droid')
## # A tibble: 6 x 14
##   name      height  mass hair_color skin_color eye_color birth_year sex
##   <chr>    <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr>
## 1 C-3PO      167    75 <NA>       gold        yellow        112 none
## 2 R2-D2       96    32 <NA>       white, bl~ red           33 none
## 3 R5-D4       97    32 <NA>       white, red red           NA none
## 4 IG-88      200   140 none       metal        red           15 none
## 5 R4-P17      96    NA none       silver, r~ red, blue      NA none
## 6 BB8        NA    NA none       none         black          NA none
## # ... with 6 more variables: gender <chr>, homeworld <chr>,
## #   species <chr>, films <list>, vehicles <list>, starships <list>
```

arrange() Example

```
## Arrange alphabetically by name in starwars
arrange(starwars, name)
## # A tibble: 87 x 14
##   name height mass hair_color skin_color eye_color birth_year sex
##   <chr> <int> <dbl> <chr> <chr> <chr> <dbl> <chr>
## 1 Ackb~ 180 83 none brown mot~ orange 41 male
## 2 Adi ~ 184 50 none dark blue NA fema~
## 3 Anak~ 188 84 blond fair blue 41.9 male
## 4 Arve~ NA NA brown fair brown NA male
## 5 Ayla~ 178 55 none blue hazel 48 fema~
## 6 Bail~ 191 NA black tan brown 67 male
## 7 Barr~ 166 50 black yellow blue 40 fema~
## 8 BB8 NA NA none none black NA none
## 9 Ben ~ 163 65 none grey, gre~ orange NA male
## 10 Beru~ 165 75 brown light blue 47 fema~
## # ... with 77 more rows, and 6 more variables: gender <chr>,
## # homeworld <chr>, species <chr>, films <list>, vehicles <list>,
## # starships <list>
```

Multiple dplyr Functions

Nest functions inside one another to perform multiple functions

```
## Select, filter, and arrange
```

```
arrange(filter(select(starwars, name, homeworld, species), species == 'Droid'))
```

```
## # A tibble: 6 x 3
```

```
##   name      homeworld species
```

```
##   <chr>    <chr>      <chr>
```

```
## 1 BB8     <NA>        Droid
```

```
## 2 C-3PO   Tatooine     Droid
```

```
## 3 IG-88   <NA>        Droid
```

```
## 4 R2-D2   Naboo       Droid
```

```
## 5 R4-P17  <NA>        Droid
```

```
## 6 R5-D4   Tatooine     Droid
```

```
## Alternative code for those functions
```

```
arrange(
  filter(
    select(starwars, name, homeworld, species),
    species == 'Droid'
  ),
  name
)
```

But either option can get very difficult to read and understand

Pipes

Pipes make a sequence of functions or operations much more readable

- Put each new step on its own line rather than all together
- Start with the first step rather than working inside-out

`x %>% f(y)` is the same as `f(x, y)`

```
## Filter with pipes
starwars %>%
  filter(species == 'Droid')
## # A tibble: 6 x 14
##   name      height  mass hair_color skin_color eye_color birth_year sex
##   <chr>    <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr>
## 1 C-3PO      167    75 <NA>      gold        yellow      112 none
## 2 R2-D2       96    32 <NA>      white, bl~ red        33 none
## 3 R5-D4       97    32 <NA>      white, red red        NA none
## 4 IG-88      200   140 none      metal       red        15 none
## 5 R4-P17      96    NA none      silver, r~ red, blue   NA none
## 6 BB8        NA    NA none      none        black       NA none
## # ... with 6 more variables: gender <chr>, homeworld <chr>,
## #   species <chr>, films <list>, vehicles <list>, starships <list>
```

Multiple dplyr Functions Using Pipes

Let's do the same sequence of three functions but using pipes

```
## Select, filter, and arrange using pipes
```

```
starwars %>%
```

```
  select(name, homeworld, species) %>%
```

```
  filter(species == 'Droid') %>%
```

```
  arrange(name)
```

```
## # A tibble: 6 x 3
```

```
##   name    homeworld species
```

```
##   <chr>   <chr>      <chr>
```

```
## 1 BB8    <NA>        Droid
```

```
## 2 C-3PO   Tatooine    Droid
```

```
## 3 IG-88   <NA>        Droid
```

```
## 4 R2-D2   Naboo       Droid
```

```
## 5 R4-P17  <NA>        Droid
```

```
## 6 R5-D4   Tatooine    Droid
```

summarize() Example

summarize() applies a function to a group of observations

- group_by() specifies the grouping to use

```
## Calculate mean height and mass by species
starwars %>%
  group_by(species) %>%
  summarize(mean_height = mean(height), mean_mass = mean(mass))

## # A tibble: 38 x 3
##   species    mean_height mean_mass
##   <chr>         <dbl>     <dbl>
## 1 Aleena           79         15
## 2 Besalisk        198        102
## 3 Cerean          198         82
## 4 Chagrian        196         NA
## 5 Clawdite        168         55
## 6 Droid           NA         NA
## 7 Dug            112         40
## 8 Ewok             88         20
## 9 Geonosian       183         80
## 10 Gungan        209.         NA
## # ... with 28 more rows
```

NA and Other Special Values

R has several special values to indicate non-standard objects or elements

- NA: Missing value
- NaN: Not a number
- NULL: “Undefined”
- Inf and -Inf: ∞ and $-\infty$

Skipping NAs

The argument `na.rm = TRUE` skips missing values

```
## Calculate non-missing mean height and mass by species
starwars %>%
  group_by(species) %>%
  summarize(mean_height = mean(height, na.rm = TRUE),
            mean_mass = mean(mass, na.rm = TRUE))

## # A tibble: 38 x 3
##   species    mean_height mean_mass
##   <chr>         <dbl>     <dbl>
## 1 Aleena           79         15
## 2 Besalisk        198        102
## 3 Cerean          198         82
## 4 Chagrian        196         NA
## 5 Clawdite        168         55
## 6 Droid           131.         69.8
## 7 Dug             112         40
## 8 Ewok             88         20
## 9 Geonosian       183         80
## 10 Gungan         209.         74
## # ... with 28 more rows
```

R Examples

OLS Regression in R

Using the `mtcars` dataset, regress `mpg` on `hp`

$$\text{mpg}_i = \beta_0 + \beta_1 \text{hp}_i + \varepsilon_i$$

Perform this simple linear OLS regression three ways:

- 1 “Canned” `lm()` function
- 2 “Hand-coded” OLS estimators
- 3 User-defined OLS function

Report parameter estimates, standard errors, t stats, and p values

But before running a regression...

Look at the mtcars Dataset

You should always double-check the structure of your dataset

```
## Look at the mtcars data
```

```
tibble(mtcars)
```

```
## # A tibble: 32 x 13
```

```
##      mpg    cyl  disp    hp  drat    wt   qsec    vs    am  gear  carb
##      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  21      6   160   110   3.9   2.62  16.5    0     1     4     4
##  2  21      6   160   110   3.9   2.88  17.0    0     1     4     4
##  3  22.8     4   108    93   3.85   2.32  18.6    1     1     4     1
##  4  21.4     6   258   110   3.08   3.22  19.4    1     0     3     1
##  5  18.7     8   360   175   3.15   3.44  17.0    0     0     3     2
##  6  18.1     6   225   105   2.76   3.46  20.2    1     0     3     1
##  7  14.3     8   360   245   3.21   3.57  15.8    0     0     3     4
##  8  24.4     4   147.    62   3.69   3.19   20      1     0     4     2
##  9  22.8     4   141.    95   3.92   3.15  22.9    1     0     4     2
## 10  19.2     6   168.   123   3.92   3.44  18.3    1     0     4     4
## # ... with 22 more rows, and 2 more variables: id <int>, ptw <dbl>
```


Summarize the mtcars Dataset

It can be helpful to generate basic summary statistics for your dataset to get a sense for the scale and variation of each variable

```
## Summarize the mtcars dataset
```

```
mtcars %>%
```

```
  select(mpg, disp, hp, wt, qsec) %>%
```

```
  summary()
```

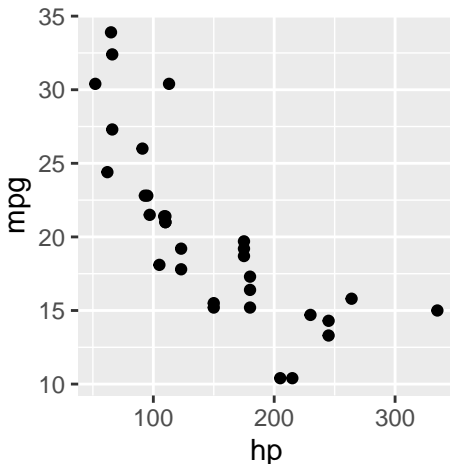
```
##           mpg           disp           hp           wt
##  Min.      :10.40   Min.      : 71.1   Min.      : 52.0   Min.      :1.513
## 1st Qu.:15.43   1st Qu.:120.8   1st Qu.: 96.5   1st Qu.:2.581
## Median :19.20   Median :196.3   Median :123.0   Median :3.325
## Mean    :20.09   Mean    :230.7   Mean    :146.7   Mean    :3.217
## 3rd Qu.:22.80   3rd Qu.:326.0   3rd Qu.:180.0   3rd Qu.:3.610
## Max.    :33.90   Max.    :472.0   Max.    :335.0   Max.    :5.424
##
##           qsec
##  Min.      :14.50
## 1st Qu.:16.89
## Median :17.71
## Mean    :17.85
## 3rd Qu.:18.90
## Max.    :22.90
```

Plot the mtcars Dataset

Plotting the data can give an idea of what to expect from your regression

```
## Plot the mtcars dataset
```

```
ggplot(data = mtcars, mapping = aes(x = hp, y = mpg)) +  
  geom_point()
```



Regression Using `lm()` Function

The `lm()` function fits a linear model to a dataset

- To see how to use the `lm()` function, type `?lm`

```
## See the help file for lm()
?lm
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

The `lm()` function requires a formula object

- $y \sim x_1 + x_2 + x_3$ regresses variable y on variables x_1 , x_2 , and x_3

Regression Using `lm()` Function

$$\text{mpg}_i = \beta_0 + \beta_1 \text{hp}_i + \varepsilon_i$$

```
## Run OLS regression
reg_lm <- lm(formula = mpg ~ hp, data = mtcars)
## Summarize OLS regression results
summary(reg_lm)
##
## Call:
## lm(formula = mpg ~ hp, data = mtcars)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -5.7121 -2.1122 -0.8854  1.5819  8.2360
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  30.09886    1.63392   18.421  < 2e-16 ***
## hp          -0.06823    0.01012   -6.742  1.79e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.863 on 30 degrees of freedom
## Multiple R-squared:  0.6024, Adjusted R-squared:  0.5892
## F-statistic: 45.46 on 1 and 30 DF,  p-value: 1.788e-07
```

Regression Using Hand-Coded Estimators

$$\text{mpg}_i = \beta_0 + \beta_1 \text{hp}_i + \varepsilon_i$$

How do we estimate the β parameters and their standard errors?

- Reminder: OLS has simple closed-form formulas!

For the general regression equation

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

we can estimate $\hat{\boldsymbol{\beta}}$ and $\widehat{\text{Cov}}(\hat{\boldsymbol{\beta}})$ using

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$

$$\widehat{\text{Cov}}(\hat{\boldsymbol{\beta}}) = s^2(\mathbf{X}'\mathbf{X})^{-1}$$

where

$$s^2 = \frac{\mathbf{e}'\mathbf{e}}{n - k}$$

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$$

Regression Using Hand-Coded Estimators

$$\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$
$$\widehat{\text{Cov}}(\hat{\beta}) = s^2(\mathbf{X}'\mathbf{X})^{-1}$$

Steps to code these estimators

- 1 Construct matrices \mathbf{X} and \mathbf{y}
- 2 Estimate parameters $\hat{\beta}$ using above equation
- 3 Calculate fitted values of \mathbf{y} , $\hat{\mathbf{y}}$
- 4 Calculate residuals, \mathbf{e}
- 5 Estimate the variance of error terms, s^2
- 6 Estimate variance-covariance matrix $\widehat{\text{Cov}}(\hat{\beta})$ using above equation
- 7 Calculate standard errors
- 8 Calculate t stats
- 9 Calculate p values
- 10 Organize results table

Regression Using Hand-Coded Estimators

Step 1: Construct matrices \mathbf{X} and \mathbf{y}

```
## Add column of ones for the constant term
reg_data <- mtcars %>%
  mutate(constant = 1)
## Select data for X and convert to a matrix
X <- reg_data %>%
  select(constant, hp) %>%
  as.matrix()
## Select data for y and convert to a matrix
y <- reg_data %>%
  select(mpg) %>%
  as.matrix()
```

Regression Using Hand-Coded Estimators

Step 1b: Make sure matrices look correct

```
## Make sure matrices look correct
```

```
head(X)
```

```
##           constant  hp
## Mazda RX4           1 110
## Mazda RX4 Wag       1 110
## Datsun 710           1  93
## Hornet 4 Drive       1 110
## Hornet Sportabout    1 175
## Valiant              1 105
```

```
head(y)
```

```
##           mpg
## Mazda RX4    21.0
## Mazda RX4 Wag 21.0
## Datsun 710    22.8
## Hornet 4 Drive 21.4
## Hornet Sportabout 18.7
## Valiant       18.1
```


Regression Using Hand-Coded Estimators

Step 2: Estimate parameters $\hat{\beta}$ using

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

```
## Estimate beta parameters
beta_hat <- solve(t(X) %*% X) %*% t(X) %*% y
beta_hat
##                mpg
## constant 30.09886054
## hp      -0.06822828
```

Regression Using Hand-Coded Estimators

Step 3: Calculate fitted values of \mathbf{y} , $\hat{\mathbf{y}}$, using

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$$

```
## Calculate fitted y values
y_hat <- X %*% beta_hat
head(y_hat)
```

##	mpg
## Mazda RX4	22.59375
## Mazda RX4 Wag	22.59375
## Datsun 710	23.75363
## Hornet 4 Drive	22.59375
## Hornet Sportabout	18.15891
## Valiant	22.93489

Regression Using Hand-Coded Estimators

Step 4: Calculate residuals, \mathbf{e} , using

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$$

```
## Calculate residuals
resid <- y - y_hat
head(resid)
##               mpg
## Mazda RX4      -1.5937500
## Mazda RX4 Wag  -1.5937500
## Datsun 710      -0.9536307
## Hornet 4 Drive  -1.1937500
## Hornet Sportabout 0.5410881
## Valiant        -4.8348913
```

Regression Using Hand-Coded Estimators

Step 5: Estimate the variance of error terms, s^2 , using

$$s^2 = \frac{\mathbf{e}'\mathbf{e}}{n - k}$$

```
## Estimate variance of error term
sigma2_hat <- t(resid) %*% resid / (nrow(X) - ncol(X))
sigma2_hat
##           mpg
## mpg 14.92248
```

Regression Using Hand-Coded Estimators

Step 6: Estimate variance-covariance matrix $\widehat{\text{Cov}}(\hat{\beta})$ using

$$\widehat{\text{Cov}}(\hat{\beta}) = s^2(\mathbf{X}'\mathbf{X})^{-1}$$

```
## Estimate variance-covariance matrix of beta estimates
vcov_hat <- c(sigma2_hat) * solve(t(X) %*% X)
vcov_hat
##           constant           hp
## constant  2.66969767 -0.0150208454
## hp        -0.01502085  0.0001024003
```

Regression Using Hand-Coded Estimators

Steps 7–9: Calculate standard errors, t stats, and p values

```
## Calculate standard errors of beta estimates
std_err <- sqrt(diag(vcov_hat))
std_err
##      constant      hp
## 1.6339210 0.0101193

## Calculate t stats of beta estimates
t_stat <- beta_hat / std_err
t_stat
##      mpg
## constant 18.421246
## hp      -6.742389

## Calculate p values of beta estimates
p_value <- 2 * pt(q = -abs(t_stat), df = nrow(X) - ncol(X))
p_value
##      mpg
## constant 6.642736e-18
## hp      1.787835e-07
```

Regression Using Hand-Coded Estimators

Step 10: Organize results table

```
## Organize regression results into matrix
results <- cbind(beta_hat, std_err, t_stat, p_value)
results
##                mpg    std_err      mpg      mpg
## constant 30.09886054 1.6339210 18.421246 6.642736e-18
## hp      -0.06822828 0.0101193 -6.742389 1.787835e-07

## Name columns of results matrix
colnames(results) <- c('Estimate', 'Std. Error', 't stat', 'p value')
results
##          Estimate Std. Error   t stat    p value
## constant 30.09886054  1.6339210 18.421246 6.642736e-18
## hp      -0.06822828  0.0101193 -6.742389 1.787835e-07
```

Regression Using Hand-Coded Estimators

Compare our hand-coded estimates to the canned `lm()` estimates

```
## Compare to lm() results
summary(reg_lm)
##
## Call:
## lm(formula = mpg ~ hp, data = mtcars)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -5.7121 -2.1122 -0.8854  1.5819  8.2360
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 30.09886    1.63392  18.421 < 2e-16 ***
## hp          -0.06823     0.01012  -6.742 1.79e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.863 on 30 degrees of freedom
## Multiple R-squared:  0.6024, Adjusted R-squared:  0.5892
## F-statistic: 45.46 on 1 and 30 DF,  p-value: 1.788e-07

results
##              Estimate Std. Error    t stat    p value
## constant 30.09886054   1.6339210 18.421246 6.642736e-18
## hp       -0.06822828   0.0101193 -6.742389 1.787835e-07
```


Regression Using User-Defined OLS Function

We want to define a new function that does the same 10 steps we just worked through

Why would we want to put these steps inside a function?

- We might want to run more than one regression
- If we define the function to take variable arguments, then we can use the same basic coding framework to run many different OLS regressions

What do we want to be the variable arguments?

- Dataset
- y variable
- x variables
- Anything else?

Regression Using User-Defined OLS Function

```
## Function to perform OLS regression
ols <- function(data, y_var, x_vars){
  ## Add column of ones for the constant term
  reg_data <- data %>%
    mutate(constant = 1)
  ## Select data for X and convert to a matrix
  X <- reg_data %>%
    select(all_of(c('constant', x_vars))) %>%
    as.matrix()
  ## Select data for y and convert to a matrix
  y <- reg_data %>%
    select(all_of(y_var)) %>%
    as.matrix()
  ## Estimate beta parameters
  beta_hat <- solve(t(X) %*% X) %*% t(X) %*% y
  ## Calculate fitted y values
  y_hat <- X %*% beta_hat
  ## Calculate residuals
  resid <- y - y_hat
  ## Estimate variance of error term
  sigma2_hat <- t(resid) %*% resid / (nrow(X) - ncol(X))
  ## Estimate variance-covariance matrix of beta estimates
  vcov_hat <- c(sigma2_hat) * solve(t(X) %*% X)
  ## Calculate standard errors of beta estimates
  std_err <- sqrt(diag(vcov_hat))
  ## Calculate t stats of beta estimates
  t_stat <- beta_hat / std_err
  ## Calculate p values of beta estimates
  p_value <- 2 * pt(q = -abs(t_stat), df = nrow(X) - ncol(X))
  ## Organize regression results into matrix
  results <- cbind(beta_hat, std_err, t_stat, p_value)
  ## Name columns of results matrix
  colnames(results) <- c('Estimate', 'Std. Error', 't stat', 'p value')
  return(results)
```

Regression Using User-Defined OLS Function

$$\text{mpg}_i = \beta_0 + \beta_1 \text{hp}_i + \varepsilon_i$$

What arguments do we need to specify?

- data, y_var, and x_vars

```
## Regress mpg on hp in mtcars dataset
ols(data = mtcars, y_var = 'mpg', x_vars = 'hp')
##           Estimate Std. Error    t stat      p value
## constant 30.09886054  1.6339210  18.421246 6.642736e-18
## hp       -0.06822828  0.0101193  -6.742389 1.787835e-07
```

We have replicated the results from `lm()` and the earlier hand-coded estimators

Regression Using User-Defined OLS Function

Now use the same function for a different regression

- Regress mpg on hp, disp, wt, qsec

```
## Regress mpg on disp, hp, wt, and qsec in mtcars dataset
ols(data = mtcars,
     y_var = 'mpg',
     x_vars = c('hp', 'disp', 'wt', 'qsec'))
```

	Estimate	Std. Error	t stat	p value
## constant	27.329637967	8.63903219	3.1635069	0.003833942
## hp	-0.018666202	0.01561305	-1.1955515	0.242266764
## disp	0.002666431	0.01073767	0.2483249	0.805762061
## wt	-4.609122617	1.26585131	-3.6411248	0.001134320
## qsec	0.544160312	0.46649316	1.1664915	0.253616070

Regression Using User-Defined OLS Function

Try a different dataset in our OLS function

- R includes a built-in dataset `iris` that includes measurements from 50 iris flowers
- Regress `Petal.Length` on `Petal.Width`, `Sepal.Length`, and `Sepal.Width`

```
## Regress Petal.Length on Sepal.Length, Sepal.Width, and Petal.Width
## in iris dataset
ols(data = iris,
     y_var = 'Petal.Length',
     x_vars = c('Petal.Width', 'Sepal.Length', 'Sepal.Width'))
```

	Estimate	Std. Error	t stat	p value
## constant	-0.2627112	0.29740608	-0.8833417	3.785039e-01
## Petal.Width	1.4467934	0.06761125	21.3987078	7.332477e-47
## Sepal.Length	0.7291384	0.05831949	12.5024834	7.656980e-25
## Sepal.Width	-0.6460124	0.06849745	-9.4311891	8.753029e-17