Introduction to R

Chase Coleman

University of Kentucky

10/10/2020

Why R?

R is an open source programming langauged, meaning it's free!

- Provide practical skills to use in either a future job or graduate school
- Get practical experience with the ecoonmics knowledge you've gained so far
- Even this presentation is made in R!
- Once you learn one langauge, easy to pick up another

Course

ECO410: R for Economists.

What to expect:

- Introduction into basic programming in R
- Be able to manage and clean data
- Program a linear regression
- Apply statistical and econometrics libraries
- Explore more advanced techniques, time permiting

Intros

Let's take a moment to introduce everyone.

Later there will be a survey to get an idea of everyones background.

Class Breakdown

The work:

- 1. Particpation/ in-class problem sets
- 2. Problem Sets
- 3. Mini Projects
- 4. Final Paper
- 5. Presentations

Syllabus

Here we take sometime to go over the syllabus

Install R and RStudio

The is done in two steps:

1) Download the latest version of R https://cran.rstudio.com
2) Download the free verison of RStudio https://rstudio.com/products/rstudio/download/

Once you've installed it we will get started with the basics!

Advice

If you do not have previous exposure to programming, it can be frustrating at times. Before we start this semester, my three tips are:

- When you don't know how to do something, look it up. A lot of coding is not memorizing code, but knowing what to look up.
- When you don't know why it's not working and you can't find an answer, check for typos. Typos can really matter a lot here [1]
- Practice. A lot of times these concepts won't make sense until you do them yourself

[1] This gets me all the time

Getting Started

- Terminology: R is the language, RStudio an integrated development environment (IDE)
- Three ways to execute R code:
  1. A script
  2. The command line
  3. R markdown/notebook (Like this presentation)

Math in R

To do Addition, Subtraction, Multiplication and Division:

```
2 + 3
```

```
[1] 5
```

```
10 - 2
```

```
[1] 8
```

```
10/5
```

```
[1] 2
```

```
2*4
```

```
[1] 8
```

Math in R

More examples: ln e

```
log(exp(1))
```

```
[1] 1
```

$\sqrt{100}$

```
sqrt(100)
```

```
[1] 10
```

$\log_{10}\${100} \$

```
log10(100)
```

```
[1] 2
```

$2^4 \$

```
2^4
```

```
[1] 16
```

Math in R

Other useful operators Comparisons: $2 < 10 $

```
2 < 10
```

```
[1] TRUE
```

The not equal to operator.

```
 5!= 5
```

```
[1] FALSE
```

The OR operator (and equal to)

```
2|50 == 2
```

```
[1] TRUE
```

The AND operator

```
2 & 50 == 40
```

```
[1] FALSE
```

Data Types

R has 6 main datatypes (we will only probably use 4)

Type	Description	Example
Numeric	This is a number that can have a decimal	$1, 14.5, 0.004$
Integer	Whole numbers	$2L, 1, -4$
Logical	Boolean values to expression true or false	TRUE, FALSE
Complex	Complex numbers	$5 + 3i$
Character	letters or word, in quotes	"a", "like this"

Examples with Data Types

```
x <- 15  # <- creates a variable
typeof(x) #   checks the type
```

```
[1] "double"
```

```
y <- 5
x+(y*y)
```

```
[1] 40
```

```
z <- 'hello '
typeof(z)
```

```
[1] "character"
```

```
print(z)    # returns the value
```

```
[1] "hello "
```

```
nchar(z)  # returns the length, space counts as a character
```

```
[1] 6
```

Examples with Data Types

What would we except doing math between a numeric and character?

Examples with Data Types

What would we except doing math between a numeric and character?

- An error. R does not support math operatios with character types (unlike a lot of programming langauges)

Data Structures/ Objects

R refers to data structures as objects.

Type	Description
Vector	A 1-D group of objects with the same type
List	A 1-D group of objects with any type
Matrix	A 2-D group of objects with the same type
Dataframe	A 2-D group of objects with any type, think excel spreadsheet
Array	A n-D group of objects with the same type

Examples of Structures

```
x <- c(1 ,2 , 3, 4 ,5)  # <- use c() to make a vector
y <- c(5:10) #< using ':' selects the whole range
z <- list('hello', 'world', c(1,2,3,4,5), FALSE) # list() for a lis

x+y
```

```
Warning in x + y: longer object length is not a multiple of shorter
length
```

```
[1]  6  8 10 12 14 11
```

Examples of Structures

Could we do $x + z$?

```
x <- c(1:5)
y <- c(5:10)
z <- list('hello', 'world', c(1,2,3,4,5), FALSE)
```

Examples of Structures

Could we do $x + z$?

```
x <- c(1:5)
y <- c(5:10)
z <- list('hello', 'world', c(1,2,3,4,5), FALSE)
```

No, because list and vector operations are not supported.

Examples of Structures

What if x and y were not the same length?

```
x <- c(1:4)
y <- c(5:10)
z <- list('hello', 'world', c(1,2,3,4,5), FALSE)
```

Examples of Structures

What if x and y were not the same length?

```
x <- c(1:4)
y <- c(5:10)
z <- list('hello', 'world', c(1,2,3,4,5), FALSE)
```

```
x+y
```

```
Warning in x + y: longer object length is not a multiple of shorter
length
```

```
[1]  6  8 10 12 10 12
```

R gives us a warning and extends the shorter one by repeating it.

Examples of Structures

Accessing within a list:

```
print(z[1])
```

```
[[1]]
[1] "hello"
```

```
print(z[3])
```

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

And from an vector:

```
print(x[1])
```

```
[1] 1
```

```
print(x[4])
```

```
[1] 4
```

Examples of Structures

Accessing within a list:

```
print(z[1])
```

```
[[1]]
[1] "hello"
```

```
print(z[3])
```

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

And from an vector:

```
print(x[1])
```

```
[1] 1
```

```
print(x[4])
```

```
[1] 4
```

Manipulating Vectors

Some operations we can do on vectors

```
x<-c(1:5)
2*x  # mulitply by a scalar
```

```
[1]  2  4  6  8 10
```

```
5 + x # add a scalar to each number in place
```

```
[1]  6  7  8  9 10
```

```
x^2 # square all numbers in place
```

```
[1]  1  4  9 16 25
```

Other ways to Generate Vectors

```
seq(2,10,by = 2) # 2:10 every 2nd number
```

```
[1]  2  4  6  8 10
```

```
rep(1,5) # 5 ones
```

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

```
a<- seq(2,10,by = 2)
a[a>4] # only picks the numbers where the condition is true
```

```
[1]  6  8 10
```

Matrices

```
a <- matrix(1:4, nrow = 2, ncol = 2) # by columns
b <- matrix(1:4, nrow = 2,  ncol = 2, byrow= TRUE) # by rows
print(a)
```

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

```
print(b)
```

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

Matrix Opertations

A bit of linear algebra

```
a + b
```

```
     [,1] [,2]
[1,]    2    5
[2,]    5    8
```

```
a * b # this does row wise multiplication
```

```
     [,1] [,2]
[1,]    1    6
[2,]    6   16
```

```
a %*% b # matrix multiplication
```

```
     [,1] [,2]
[1,]   10   14
[2,]   14   20
```

Dataframes

Let's look at a built in dataframe, mtcars to explore how dataframes works

```
df = mtcars
head(df) # shows the top of the df
```

```
                   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
```

Dataframes: Selecting Data

Access the data using indexing

```
df[1:2,]  # selects the first two rows
```

```
                mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4        21   6  160 110  3.9 2.620 16.46  0  1    4    4
Mazda RX4 Wag    21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

Dataframes: Selecting Data

Access the data using indexing

```
df[1:2,]  # selects the first two rows
```

```
                mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4        21   6  160 110  3.9 2.620 16.46  0  1    4    4
Mazda RX4 Wag    21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

```
df[,1]  # selects the first column
```

```
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.
[16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.
[31] 15.0 21.4
```

Dataframes: Selecting Data

Access the data using indexing

```
df[1:2,]  # selects the first two rows
```

```
                mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21   6  160 110  3.9 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

```
df[,1]  # selects the first column
```

```
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.
[16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.
[31] 15.0 21.4
```

```
df[3,1]  # selects the 3 row, first column
```

```
[1] 22.8
```

Dataframes: Selecting Data

You can also use Row and column names

```
df["Ferrari Dino","mpg"]
```

```
[1] 19.7
```

Dataframes: Selecting Data

You can also use Row and column names

```
df["Ferrari Dino","mpg"]
```

```
[1] 19.7
```

Or if you want to know about the size:

```
dim(df) # dimension in row X col
```

```
[1] 32 11
```

```
nrow(df) # gives number of rows
```

```
[1] 32
```

```
ncol(df) # gives number of cols
```

```
[1] 11
```

Conditional Statement & Functions

- if, else
- repeat
- while loops
- for loops

if/else statement

```
x<- -5
if ( x > 0) {   # condition goes in (), statement in {}
print('x is a positive number')
} else if (x== 0) {  # else if is an additional condition
    print(' x is zero')
  } else {  #  if the first two conditions are not met
print('x is a negative number' )
}
```

```
[1] "x is a negative number"
```

Functions

Functions in R work just like functions in Math!

Given input(s), functions produce output(s)

Functions

Functions in R work just like functions in Math!

Given input(s), functions produce output(s)

```
isPositive<- function(x){ # the input goes here with function
ifelse(x > 0, TRUE, FALSE)
}
isPositive(x)
```

```
[1] FALSE
```

Repeat Loops

Repeat loops repeat until they reach a 'break' [2]. That means it could go on forever if there is no break.

```
repeat{
  x = x+1
  y = isPositive(x)
  print(x)
  if(y == TRUE){ # will repeat until x is positive
    break
  }

}
```

```
[1] -4
[1] -3
[1] -2
[1] -1
[1] 0
[1] 1
```

[2] both *break* and *next*

While Loops

While loops continue until the condition is met or it reachs a break.

```
x<- 5
while( isPositive(x)){  # the condition is in ()
  x = x - 1
  print(x)
}
```

```
[1] 4
[1] 3
[1] 2
[1] 1
[1] 0
```

For Loops

For each value in a sequence, the statement repeats.

```
x<- c(1:10)
for(number in x){ # for each number in x it will do the statement
  x[number] = number + number
}
print(x)
```

```
 [1]  2  4  6  8 10 12 14 16 18 20
```

Note: A faster and better implentation would just be $x = x + x$ or $x = 2x$

Word of Caution

While R has the ability to do loops, it is much faster if you can avoid them. Also while and repeat loops run into the potential problem of infinite loop (never reaches break or meets condition).