Introduction to R

Mauricio Romero
(Based on Nick C. Huntington-Klein's notes)

Getting familiar with R

- R is a language for working with data

- RStudio is an environment for working with that language

- Excel/Sheets is a great tool for accountants, not for working with data

- Learning to program is a highly valuable skill (regardless of what you want to do)

- Programming: a language to communicate with the computer

- Programming requires you to be very precise: Computer will do exactly as told
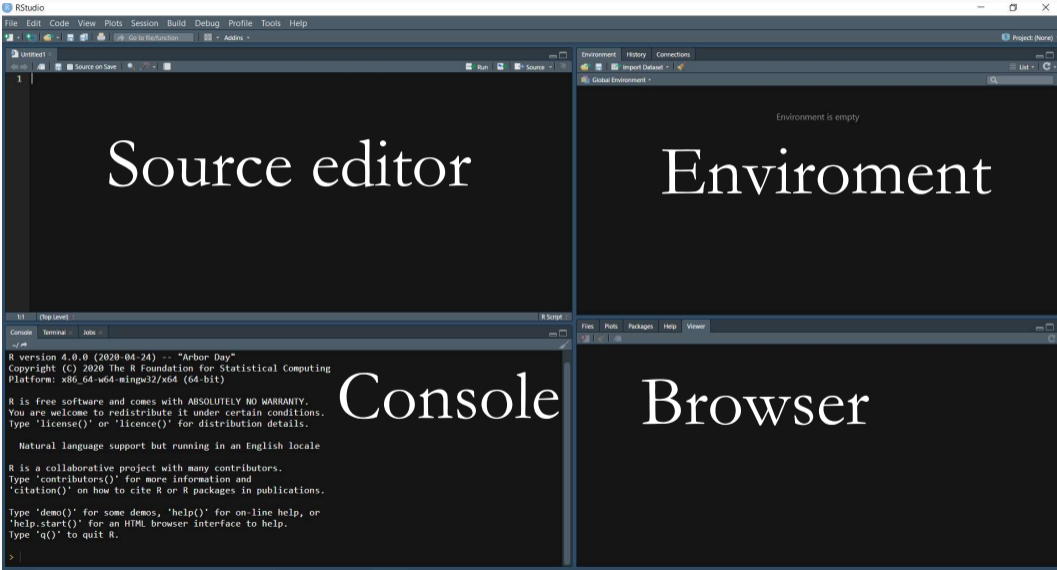
RStudio – The Basics

Objects and Functions

Introduction to R

RStudio – The Basics

Objects and Functions

RStudio Panes

- Console

- Environment Pane

- Browser Pane

- Source Editor

Console

- Typically bottom-left

- You can type in code and have it run immediately

- It will also show any output or errors

Let's copy/paste some code in there to run

```
#Generate 500 heads and tails
data <- sample(c("Heads","Tails"),500,replace=TRUE)
#Calculate the proportion of heads
mean(data=="Heads")
#This line should give an error - it didn't work!
data <- sample(c("Heads","Tails"),500,replace=BLUE)
#This line should give a warning
#It did SOMETHING but maybe not what you want
mean(data)
#This line won't give an error or a warning
#But it's not what we want!
mean(data=="heads")
```

Console

- See the code that we've run

- See the output of that code, if any

- See any errors or warnings (in **red**)

- Errors mean it didn't work

- Warnings mean it **maybe** didn't work.

- Just because there is no error or warning does **not** mean it **did** work! Always check

RStudio Panes

```
> #Generate 500 heads and tails
> data <- sample(c("Heads","Tails"),500,replace=TRUE)
> #Calculate the proportion of heads
> mean(data=="Heads")
[1] 0.48
> #This line should give an error - it didn't work!
> data <- sample(c("Heads","Tails"),500,replace=BLUE)
Error in sample.int(length(x), size, replace, prob) :
  object 'BLUE' not found
> #This line should give a warning
> #It did SOMETHING but maybe not what you want
> mean(data)
[1] NA
Warning message:
In mean.default(data) : argument is not numeric or logical: returning NA
> #This line won't give an error or a warning
> #But it's not what we want!
> mean(data=="heads")
[1] 0
>
```

Environment pane

- Typically is on the top-right
- Two important tabs: Environment and History
- History: Log of what we have done
  - Can re-run commands by double-clicking them or hitting Enter
  - Send to console with double-click/enter
  - Send to source pane with Shift+double-click/Enter
  - Or use "To Console" or "To Source" buttons
- Environment: Objects we have created
  - All the objects we have in memory
  - For example, we created the "data" object, so we can see that in Environment
  - It shows us lots of useful information about that object too (e.g., size)
  - You can erase everything with the little broom (equivalent to "rm(list=ls())")

Browser Pane

- Typically bottom-right

- Mostly, **outcome** of what you do will be seen here

Browser Pane: Files Tab

- Files Tab
    - Basic file browser
    - Handy for opening up files
    - Can also help you set the working directory (same as "setwd(file_path)")
        - Go to folder
        - In menu bar, Session
        - Set Working Directory
        - To Files Pane Location

Browser Pane: Plots and Viewer tabs

- When you create something that must be viewed, like a plot, it will show up here
- For example:

```
data(LifeCycleSavings)
plot(density(LifeCycleSavings$pop75),
main='Percent of Population over 75')
```

- For example (using ggplot):

```
data(LifeCycleSavings)
library(ggplot2)
ggplot(LifeCycleSavings,aes(x=pop75))+
stat_density(geom='line')+
ggtitle('Percent of Population over 75')
```

- "Export" button here - save plots you've created (better to do this via code)

Percent of Population over 75
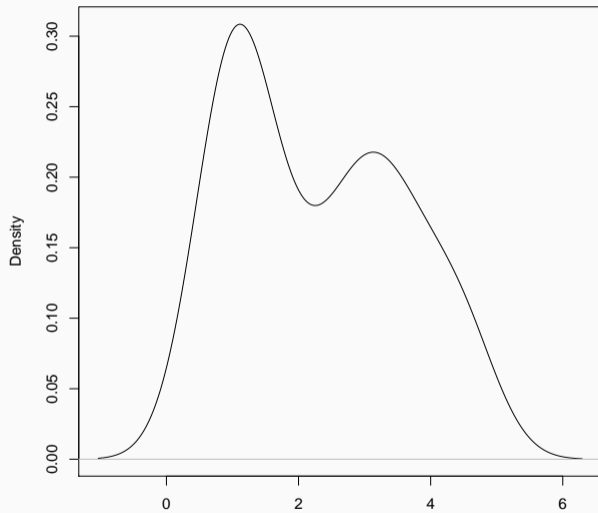
N = 50   Bandwidth = 0.5312

Browser Pane: Packages Tab

- Install new packages and load them in

- We'll be talking more about packages later

- I generally avoid this tab; better to do this via code

  - Why? Replicability!

  - A **VERY** important reason to use code and not the GUI or Excel

  - Make sure your future self (or someone else) knows how to use your code

- "Update" button (check for package updates)

Browser Pane: Help Tab

- This is where help files appear when you ask for them

- You can use the search bar , or "help()" in the console

- More on this later

Source Pane

- Top left

- You should be working with code FROM THIS PANE, not the console!

  - Why? Replicability!

  - Also, COMMENTS! USE THEM! PLEASE! "#" lets you write a comment

- Switch between tabs like a internet browser

Running Code from the Source Panel

- Select a chunk of code and hit the "Run" button (ctrl+enter)

- Going one line at a time lets you check for errors more easily

Running Code from the Source Panel

```
data(mtcars)
mean(mtcars$mpg)
mean(mtcars$wt)
372+565
log(exp(1))
2^9
(1+1)^9
```

```
> data(mtcars)
> mean(mtcars$mpg)
[1] 20.09062
> mean(mtcars$wt)
[1] 3.21725
> 372+565
[1] 937
> log(exp(1))
[1] 1
> 2^9
[1] 512
> (1+1)^9
[1] 512
>
```

Autocomplete

- RStudio comes with autocomplete!

- Typing in the Source Pane or the Console, it will try to fill in things for you

  - Command names (shows the syntax of the function too!)

  - Object names from your environment

  - Variable names in your data

- Try redoing the code we just did, typing it out

Help

- Autocomplete is one way that RStudio tries to help you out

- The way that R helps you out the most is with the documentation

- When you start doing serious programming, the most important skills are:

  - Knowing to read documentation

  - Knowing to search the internet for help

Help

- You can get the documentation on most R objects using the "help()" function

- "help(mean)", for example, will show you:

  - What the function is

  - The syntax for the function (i.e., what it takes and what it spits out)

  - The available options for the function

  - Other, related functions, like "weighted.mean"

  - Ideally, some examples of proper use

- Not just for functions/commands - some data sets too! Try "help(mtcars)"

Searching the Internet

- Professional programmers spend a lot of time looking up how to do stuff online

- Just Google (or whatever) what you need! There will usually be a resource

- R-bloggers, Quick-R, StackOverflow

- Or just Google. Try including "R" and the name of what you want in the search

- If "R" isn't turning up anything, try "Rstats" or "R Project"

- Ask on Twitter with '#rstats'

- Ask on StackExchange or StackOverflow

Searching the Internet

- We've used data sets LifeCycleSavings and mtcars with the data() function

- What data sets can we get this way?

- Let's try "help(data)"

- Let's try searching the internet

Introduction to R

RStudio – The Basics

Objects and Functions

RStudio – The Basics

Objects and Functions

R is an object oriented programming language

- What can you do in R?

  - Create objects

  - Look at objects

  - Manipulate objects

- That's it. That's all. That's what R does.

Creating a Basic Object

- Let's create an object. Do this with the assignment operator "<-" (a.k.a., "gets")

  ```
  a <- 4
  ```

- Creates an object called 'a'
  - What is that object? It's a number
  - Specifically, it's the number 4
  - We know that because we took that 4 and we shoved it into 'a'
- Why store it as an object? To look at it and manipulate it
- We can do more complex calculations before storing it, too.

  ```
  b <- sqrt(16)+10
  ```

Looking at Objects

- We can see this object that we've created in the Environment pane

- Putting that object on a line by itself in R will show us what it is
  ```
  a
  b
  ```

```
> a <- 4
> b <- sqrt(16)+10
> a
[1] 4
> b
[1] 14
> |
```

Looking at Objects

- We can create an object and look at it in the console without storing it

  ```
  3
  a+b
  ```

- We can run objects through **functions** to look at them in different ways

  ```
  #What does a look like if we take the square root of it?
  sqrt(a)
  #What does it look like if we add 1 to it?
  a + 1
  #If we look at it, do we see a number?
  is.numeric(a)
  ```

```
> 3
[1] 3
> a+b
[1] 18
> #What does a look like if we take the square root of it?
> sqrt(a)
[1] 2
> #What does it look like if we add 1 to it?
> a + 1
[1] 5
> #If we look at it, do we see a number?
> is.numeric(a)
[1] TRUE
>
```

Manipulating Objects

```
#Looked like with 1 added, but a wasn't changed
a
#Let's save a+1 as something else
b <- a + 1
#And let's overwrite a with its square root
a <- sqrt(a)
a
b
```

```
[1] TRUE
> #Looked like with 1 added, but a wasn't changed
> a
[1] 4
> #Let's save a+1 as something else
> b <- a + 1
> #And let's overwrite a with its square root
> a <- sqrt(a)
> a
[1] 2
> b
[1] 5
> |
```

Some Notes

- Even though we changed 'a', 'b' was already set using the old value of 'a',

    - So was still '4+1=5', not '2+1=3'

- 'a' basically got **reassigned** with '<-'. That's how we got it to be '2'

- Everything in R is just manipulating objects but with more complex objects and more complex functions!

Types of Objects

- We already determined that 'a' was a number

- What else could it be? What other kinds of variables are there?

- Some basic object types:

  - Numeric: A single number

  - Character: A string of letters, like ''hello''

  - Logical: 'TRUE' or 'FALSE' (or 'T' or 'F')

  - Factor: A category, like ''left handed', 'right handed', or 'ambidextrous''

  - Vector: A collection of objects of the same type

Characters

- A character object is a piece of text, held in quotes like '' '' or ''' '''

- For example, maybe you have some data on people's addresses
  ```
  address <- "321 Fake St."
  address
  is.character(address)
  ```

```
> address <- "321 Fake St."
> address
[1] "321 Fake St."
> is.character(address)
[1] TRUE
> |
```

Logical

- Logicals are binary: TRUE or FALSE. Lots of data is binary

```
c <- TRUE
is.logical(c)
is.character(a)
is.logical(is.numeric(a))
```

- Logicals are used a lot in programming too to evaluate whether a conditions hold

```
a > 100
a > 100 | b == 5
```

- '&' is AND
- '|' is OR
- To check equality use '==', not '='
- '>=' is greater than OR equal to, similarly for '<='

```
> c <- TRUE
> is.logical(c)
[1] TRUE
> is.character(a)
[1] FALSE
> is.logical(is.numeric(a))
[1] TRUE
> a > 100
[1] FALSE
> a > 100 | b == 5
[1] TRUE
>
```

- They are also equivalent to 'TRUE=1' and 'FALSE=0' which comes in handy

- We can use 'as' functions to change object type (although for 'T=1, F=0' it does it automatically)

```
as.numeric(FALSE)
TRUE + 3
```

```
> as.numeric(FALSE)
[1] 0
> TRUE + 3
[1] 4
> |
```

Let's think about what these lines might do

```
is.logical(is.numeric(FALSE))
is.numeric(2) + is.character('hello')
is.numeric(2) & is.character(3)
TRUE | FALSE
TRUE & FALSE
```

```
> is.logical(is.numeric(FALSE))
[1] TRUE
> is.numeric(2) + is.character('hello')
[1] 2
> is.numeric(2) & is.character(3)
[1] FALSE
> TRUE | FALSE
[1] TRUE
> TRUE & FALSE
[1] FALSE
```

Factors

- Factors are categorical variables (i.e., mutually exclusive groups)

- They look like strings, but they're more like logicals with more than two levels

- Factors have **levels** showing the possible categories you can be in
  ```
  e <- as.factor('left-handed')
  levels(e) <- c('left-handed','right-handed','ambidextrous')
  e
  ```

```
> e <- as.factor('left-handed')
> levels(e) <- c('left-handed','right-handed','ambidextrous')
> ee <- as.factor('left-handed')
> levels(e) <- c('left-handed','right-handed','ambidextrous')
> e
[1] left-handed
Levels: left-handed right-handed ambidextrous
> |
```

Vectors

- Data is basically a bunch of variables all put together

- A lot of R works with vectors, which are a bunch of objects all put together!

- Use 'c()' (concatenate) to put objects of the same type together in a vector

- Use square brackets to pick out parts of the vector
  ```
  d <- c(5,6,7,8)
  c(is.numeric(d),is.vector(d))
  d[2]
  ```

```
> d <- c(5,6,7,8)
> c(is.numeric(d),is.vector(d))
[1] TRUE TRUE
> d[2]
[1] 6
>
```

Vectors

- Statistics helps us make sense of lots of different measurements of the same thing

- Thus, lots of statistical functions look at multiple objects

```
mean(d)
c(sum(d),sd(d),prod(d))
```

```
> mean(d)
[1] 6.5
> c(sum(d),sd(d),prod(d))
[1]    26.000000     1.290994 1680.000000
> 
```

Vectors

- We can perform the same operation on all parts of the vector at once!

```
d + 1
d + d
d > 6
```

```
> d + 1
[1] 6 7 8 9
> d + d
[1] 10 12 14 16
> d > 6
[1]  FALSE FALSE  TRUE  TRUE
```

Vectors

- Factors make a lot more sense as a vector

```
continents <- as.factor(c('Asia','Asia','Asia',
                          'N America','Europe',
                          'Africa','Africa'))
table(continents)
continents[4]
```

```
> continents <- as.factor(c('Asia','Asia','Asia',
+                           'N America','Europe',
+                           'Africa','Africa'))
> table(continents)
continents
   Africa      Asia    Europe N America
        2         3         1         1
> continents[4]
[1] N America
Levels: Africa Asia Europe N America
>
```

Value matching

- Create logicals seeing if a value matches ANY value in a vector with '%in%'

```
3 %in% c(3,4)
c('Nick','James') %in% c('James','Andy','Sarah')
```

```
> 3 %in% c(3,4)
[1] TRUE
> c('Nick','James') %in% c('James','Andy','Sarah')
[1] FALSE  TRUE
>
```

Some basic vector manipulations

```
1:8
rep(4,3)
rep(c('a','b'),4)
numeric(5)
character(6)
sample(1:20,3)
sample(c("Heads","Tails"),6,replace=TRUE)
```

```
> 1:8
[1] 1 2 3 4 5 6 7 8
> rep(4,3)
[1] 4 4 4
> rep(c('a','b'),4)
[1] "a" "b" "a" "b" "a" "b" "a" "b"
> numeric(5)
[1] 0 0 0 0 0
> character(6)
[1] "" "" "" "" "" ""
> sample(1:20,3)
[1]  2 11  3
> sample(c("Heads","Tails"),6,replace=TRUE)
[1] "Heads" "Heads" "Heads" "Tails" "Tails" "Heads"
> |
```

Vector Test

- If we do 'f <- c(2,3,4,5)', then what will the output of these be?

```
f^2
f + c(1,2,3,4)
c(f,6)
is.numeric(f)
mean(f >= 4)
f*c(1,2,3)
length(f)
length(rep(1:4,3))
f/2 == 2 | f < 3
as.character(f)
f[1]+f[4]
c(f,f,f,f)
f[f[1]]
f[c(1,3)]
f %in% (1:4*2)
```

```
> f <- c(2,3,4,5)
> f^2
[1]  4  9 16 25
> f + c(1,2,3,4)
[1] 3 5 7 9
> c(f,6)
[1] 2 3 4 5 6
> is.numeric(f)
[1] TRUE
> mean(f >= 4)
[1] 0.5
> f*c(1,2,3)
[1]  2  6 12  5
Warning message:
In f * c(1, 2, 3) :
  longer object length is not a multiple of shorter object length
> length(f)
[1] 4
> length(rep(1:4,3))
[1] 12
> f/2 == 2 | f < 3
[1]  TRUE FALSE  TRUE FALSE
> as.character(f)
[1] "2" "3" "4" "5"
> f[1]+f[4]
[1] 7
> c(f,f,f,f)
 [1] 2 3 4 5 2 3 4 5 2 3 4 5 2 3 4 5
> f[f[1]]
[1] 3
> f[c(1,3)]
[1] 2 4
```

Homework

- Create a factor that randomly samples six ''Male'' or ''Female'' people.

- Add up all the numbers from 18 to 763, then get the mean

- What happens if you make a list with a logical, a numeric, AND a string in it?

- Figure out how to use `paste0()` to turn `c('a','b')` into ''ab''

- Use `[]` to turn `h <- c(10,9,8,7)` into `c(7,8,10,9)` and call it `j`

- (Several ways) Create a vector with eleven 0's, then a 5.

- Use `floor()` or `%%` to count how many multiples of 4 between 433 and 899