

Lecture 1: An Introduction & Motivation for R Programming

Harris Coding Camp – Standard Track

Summer 2022

Welcome to Coding Camp!

- ▶ Why are we here?
- ▶ What are we going to do?
- ▶ A quick introduction to R and RStudio

Teaching Members

- ▶ Instructors
 - ▶ Standard Track: Arthur Cheib, Sheng-Hao Lo
 - ▶ Accelerated Track: Ari Anisfeld
- ▶ Head TA: Rubina Hundal
 - ▶ All logistics issues
- ▶ You will also have several TAs who will be helping you along the way!
 - ▶ TA sessions
 - ▶ Canvas Discussion Board

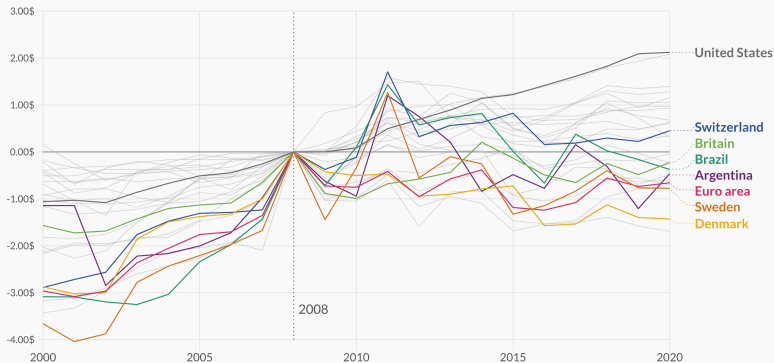
Why learn coding?

- ▶ Computation is an essential party of modern-day applied statistics and *quantitative* policy analysis
- ▶ Many public policy jobs and the Harris curriculum rely on programming
 - ▶ to quickly engage with policy data
 - ▶ to complete statistical analyses
- ▶ Examples
 - ▶ Change in test scores among different groups due to online education
 - ▶ Change in unemployment rate due to CARES Act
 - ▶ Change in expenses of a certain good compared to a specific time period

An Example

Compared to the financial crisis in 2008, how much more or less do you have to pay for a Big Mac today?

The index chart visualizes the price changes (in USD) of a Big Mac based on a 2008 as index year. The **Big Mac Index** is published by The Economist as an informal way to provide a test of the extent to which market exchange rates result in goods costing the same in different countries. It seeks to *make exchange-rate theory a bit more digestible* and takes its name from the Big Mac, a hamburger sold at McDonald's restaurants.



Visualization by Cédric Scherer • Data by The Economist • The index chart shows the 27 countries that provide Big mac prices for all years from 2000 to 2020. In case a country was reported twice per year, the mean value was visualized.

Why R?

- ▶ R is an extremely powerful programming language and statistical software environment
 - ▶ Great data manipulation and visualization suite
 - ▶ Strong statistical packages (e.g. program evaluation, machine learning)
- ▶ Complete programming language with low barriers to entry
- ▶ Open source and free

A bit More about R

- ▶ We will use R for the entire Stats sequence in Fall and Winter
- ▶ We will use R through RStudio, which is a more user-friendly interface that “sits atop” R
- ▶ In order to use RStudio, you must also have both RStudio and R installed on your computer (R is the “engine” for RStudio)
- ▶ When working in RStudio, you can either:
 - ▶ Work **interactively** by entering each line of code into the console
 - ▶ Write a **script** (a .R file) in which you save your code in a file and submit multiple lines of code at once

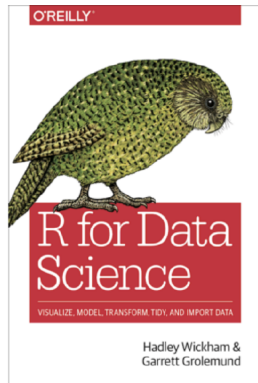
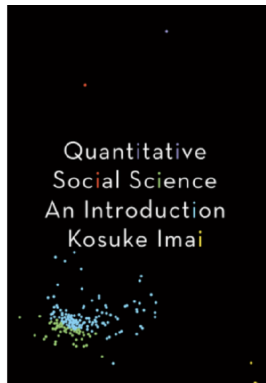
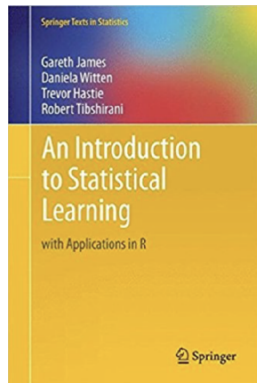
What will we cover?

0. Motivation/Installation of R
 1. Installing Packages and Reading Data
 2. Basic Data Manipulation and Analysis
 3. Data Visualization
 4. More on Data Manipulation
 - ▶ Grouped Analysis, Iteration, Functions
-
- ▶ In Stats 1/2 and other courses, you will build off of these lessons:
 - ▶ extend your capabilities with the functions we teach you
 - ▶ introduce statistics functions
 - ▶ introduce new packages and tools based on needs

Learning philosophy

- ▶ We learn coding by experimenting with code
- ▶ Coding requires a different modality of thinking
- ▶ Coding can be frustrating
- ▶ We develop self-sufficiency by learning where to get help and how to ask for help
- ▶ Coding lab is for you
- ▶ This is just the beginning of your programming journey!

Textbooks and Resources



- ▶ Get situated with R for Data Science <https://r4ds.had.co.nz/>

Textbooks and Resources

How to actually learn any new programming concept



Essential

Changing Stuff and
Seeing What Happens

O RLY?

@ThePracticalDev

The internet will make those bad words go away



Essential

Googling the
Error Message


O RLY?

The Practical Developer
@ThePracticalDev

- ▶ Google is your friend for idiosyncratic problems

Textbooks and Resources

Cutting corners to meet arbitrary management deadlines



Essential

Copying and Pasting from Stack Overflow

O'REILLY*

*The Practical Developer
@ThePracticalDev*

stackoverflow

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

Teams

Q&A for work

Learn More

Search...

What are the differences between "=" and "<=" in R?

582

I know that operators are slightly different, as this example shows

```
x <- y <- 5
x = y = 5
x <= y <= 5
x = y = 5
# Error in (x <= y) = 5 : could not find function "<=="
```

206

But is this the only difference?

assignment-operator

share improve this question

asked Mar 8 '18 at 9:30

answered Nov 19 '18 at 12:14

27

As noted [here](#) the origins of the <= symbol come from old APL keyboards that actually had a single <= key on them. — joran Dec 12 '14 at 17:30

add a comment

7 Answers

active oldest votes

568

The difference in [assignment operators](#) is clearer when you use them to set an argument value in a function call. For example:

```
median(x = 1:10)
# Error: object 'x' not found
```

In this case, `x` is declared within the scope of the function, so it does not exist in the user workspace.

```
median(x <- 1:10)
# [1] 1 2 3 4 5 6 7 8 9 10
```

In this case, `x` is declared in the user workspace, so you can use it after the function call has been completed.

There is a general preference among the R community for using <= for assignment (other than in

► Stack Overflow is your another friend!

How will we progress?

1. Live lectures:
 - ▶ Focus on main idea first
 - ▶ Try it yourself – you learn coding by coding! (work on short problems with TA support)
2. Practice in TA sessions (Most important part!):
 - ▶ Again – you learn coding by coding!
 - ▶ Break up into small groups and work on problems with peer and TA support
3. Additional help:
 - ▶ Send emails to Head TA for logistics issues
 - ▶ Post questions to Canvas Discussion Board (Teaching team will monitor and reply)
4. Final project:
 - ▶ It's optional; more details on next slide

Final project (optional)

You'll know you're ready for policy school coding, if you can open a data set of interest to you and produce meaningful analysis. For the final project, you will:

- ▶ Pick a data set aligned with your policy interests (or not)
- ▶ Use programming skills to engage with data and make a data visualization showing something you learned from the data

A quick introduction to R and R Studio

We will

- ▶ Discuss what RStudio is
- ▶ Introduce minimal information to get started working with R
- ▶ Expose you to the different data types in R
- ▶ Learn different types of operators

What is RStudio?

R Studio is an “integrated development environment” for R.

- ▶ It provides a console to access R directly
- ▶ A text editor to write R scripts and work with Rmds
- ▶ An environment and history tab that provide useful information about what objects you have in your R session
- ▶ A help / plots / files / packages etc. section

Let's take a tour in RStudio!

RStudio Layout

1. SOURCE

This is where you write your code!

Your code will not be evaluated until you "Run" them to the console.

Click "Run" to send your code to the console

2. CONSOLE

This is where your code from the Source is evaluated by R.

You can also use the console to perform quick calculations that you don't need to save

3. Environment / History

Here you can see what objects are in your working space (Environment) or view your command history (History)

4. Files / Plots / Packages / Help

Here you can see file directories, view plots, see your packages, and access R Help

Anatomy of RStudio

The screenshot shows the RStudio interface with several panels and annotations:

- Source Editor (Top Left):** Contains R code for loading data and creating variables. A callout bubble points to this panel: "This window is a 'script'. Here's where you write and edit your programs."
- Environment Panel (Top Right):** Shows the current environment with variables like 'data', 'df1', 'df2', and 'df3'. A callout bubble points to this panel: "Your data lives here."
- Files Panel (Bottom Left):** Shows the file explorer. A callout bubble points to this panel: "You'll find your files, plots, packages and other stuff over here."
- Console/Terminal (Bottom Center):** Shows the output of the R code. A callout bubble points to this panel: "Down here is the console. This is where your code runs and where R gets its work done."
- Help Panel (Bottom Right):** Shows the documentation for the 'ifelse' function. A callout bubble points to this panel: "You'll find your files, plots, packages and other stuff over here."

The R code in the Source Editor is as follows:

```
1 setwd("~/RStudioTools")
2 library(tidyverse)
3 library(data.table)
4 read_csv("data.csv")
5 df <- read_csv("data.csv", col_names = FALSE)
6 glimpse(df)
7
8
9 df <- df %>%
10   mutate("Treatment" = "...") %>%
11   mutate("Subject" = "...") %>%
12   mutate("Task" = "...") %>%
13   mutate("Time" = "...") %>%
14   mutate("Choice" = "...") %>%
15   mutate("Reward" = "...") %>%
16   mutate("Trial" = "...") %>%
17   mutate("Day" = "...") %>%
18
19 df3 <- df %>%
20   filter(Choice != 0)
21 df3 <- df3 %>%
22   no.unit(Choice)
23
24 df <- df %>% #this block isn't working!
25   auto_clean = 1 %>%
26   group_by(Subject) %>%
27   summarise(Subtotal = sum(Total)) %>%
28   filter(Total > 0)
```

The console output shows the R version and the RStudio logo.

Basic syntax: Variable assignment

We can think of a variable as a *container* with a name, such as

- x
- stats_score
- harris_gpa_average

Each container can contain *one or more* values (more to come in Lecture 3)

Basic syntax: Variable assignment

We can create a new variable and assign a value to it using = or <-.

```
x = 7  
x
```

```
## [1] 7
```

In the above bit of code, we created a variable called x and stored the value 7 inside of it.

Basic syntax: Variable assignment

We can create a new variable and assign a value to it using `=` or `<-`.

```
my_number <- 3  
my_number
```

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

In the above bit of code, we created a variable called `my_number` and stored the value 3 inside of it.

Basic syntax: Variable assignment

We can treat our variable like a regular number, and do arithmetic with it:

```
my_number <- 3  
my_number
```

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

```
my_output <- 12 * my_number + 2  
my_output
```

```
## [1] 38
```

Basic syntax: Variable assignment

Variables can be reassigned if needed:

```
my_number <- 4  
my_number # Now it becomes 4
```

```
## [1] 4
```

```
my_output <- 12 * my_number + 2  
my_output # Its value changes accordingly
```

```
## [1] 50
```


Basic syntax: Add comments

- ▶ We can add comments to our code using the # character
- ▶ It is useful to document our code in this way so that others (and us the next time we read it) have an easier time following what the code is doing
- ▶ Anything after # is ignored by R when executes code

```
#my_number is the variable I set equal to 4  
my_number - 10 # This should be equal to -6
```

```
## [1] -6
```

Functions

Functions are procedures that take an input and provide an output.
Let's see some already defined functions:

```
x <- sqrt(4)
x
```

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

```
y <- mean(c(3, 4, 5, 6, 7))
y
```

```
## [1] 5
```

- ▶ We can combine values/objects in a new object with the function `c()` (c for combine). When objects are combined, they are called a *vector* (more details later)
- ▶ We can also define new functions by ourselves (more details later)

Using R as a calculator

$+$, $-$, $*$, and $/$. Also, $^$ (Exponent).

```
7 + 5
```

```
## [1] 12
```

```
(4 + 6) * 3 - 2
```

```
## [1] 28
```

```
7 / 5
```

```
## [1] 1.4
```

```
2^4
```

```
## [1] 16
```

Using R as a calculator

- ▶ R has many built-in mathematical functions
- ▶ To call a function, we type its *name*, followed by parentheses
- ▶ Anything we type inside the parentheses is called the function's *arguments*

```
sin(1)    # trigonometric functions
```

```
## [1] 0.841471
```

```
log(1)    # natural logarithm
```

```
## [1] 0
```

```
exp(0.5)  #  $e^{(1/2)}$ 
```

```
## [1] 1.648721
```

- ▶ Typing a ? before the name of a function will open the help page for that function

Detour: Executing commands in R

Three ways to execute commands in R:

1. Type/copy commands directly into the console
2. R scripts (.R files)
 - ▶ This is just a text file full of R commands
 - ▶ Can execute one command at a time, several commands at a time, or the entire script
3. 'code chunks' in RMarkdown (.Rmd files)
 - ▶ Can execute one command at a time, one chunk at a time, or "knit" the entire document
 - ▶ More on this later

Operators can return special values

Inf is infinity. You can have either positive or negative infinity.

```
1 / 0
```

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

```
-5 / 0
```

```
## [1] -Inf
```

NaN means Not a Number. It's an undefined value.

```
0 / 0
```

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

Try it yourself

1. $30 + 6 \times 5^8 - \log(50) = ?$

2. $568 \times \frac{135}{\log(1)} = ?$

3. $\frac{15^0 - 1}{\sin(0)} = ?$

Logical Operations

operator	definition	operator	definition
<	less than	<code>x y</code>	x OR y
<=	less than or equal to	<code>is.na(x)</code>	test if x is NA
>	greater than	<code>!is.na(x)</code>	test if x is not NA
>=	greater than or equal to	<code>x %in% y</code>	test if x is in y
==	exactly equal to	<code>!(x %in% y)</code>	test if x is not in y
!=	not equal to	<code>!x</code>	not x
<code>x & y</code>	x AND y		

Comparison Operators

These are also binary operators; they take two objects, and give back a *Boolean*

```
7 > 5 # greater than
```

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

```
7 < 5 # less than
```

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

```
7 >= 5 # greater than or equal to
```

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

Comparison Operators

```
7 <= 5 # less than or equal to
```

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

```
7 == 5 # equality (two equals signs, read as "is equal to")
```

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

```
7 != 5 # inequality (read as "is not equal to")
```

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

Reminder: `==` is a comparison operator, `=` is an assignment operator

& (and)

- ▶ TRUE & TRUE -> TRUE
- ▶ TRUE & FALSE -> FALSE
- ▶ FALSE & TRUE -> FALSE
- ▶ FALSE & FALSE -> FALSE

```
(5 < 7) & (6 * 7 == 42)
```

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

```
(5 < 7) & (6 * 7 < 42)
```

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

```
(5 > 7) & (6 * 7 == 42)
```

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

| (or)

- ▶ TRUE | FALSE -> TRUE
- ▶ FALSE | TRUE -> TRUE
- ▶ TRUE | TRUE -> TRUE
- ▶ FALSE | FALSE -> FALSE

```
(5 < 7) | (6 * 7 < 42)
```

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

```
(5 > 7) | (6 * 7 == 42)
```

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

```
(5 < 7) | (6 * 7 == 42)
```

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

Try it yourself

Guess the output of the following codes, and then run the codes to check your answer:

```
x <- 6  
(x < 9) & (x > 3)  
(x < 9) | (x > 7)  
(x > 8) | (x > 9)
```

```
x = 20  
y = 30  
(x == 20) & (y == 30)  
(x == 20) | (y == 50)  
(x + 5^{8} - log(50) < 2000000) | (3*y - log(500) == 0)
```

Recap

Now you should understand how to:

- ▶ navigate and use RStudio's features
 - ▶ particularly, the console, the text editor and help
- ▶ assign objects to names with `<-` or `=`
- ▶ learn basic syntax
 - ▶ Variable assignment
 - ▶ Add comments
 - ▶ Built-in Functions
 - ▶ Logical Operators