Introduction to R for Statistics

Text and code written by Laura K. Nelson (l.nelson@northeastern.edu)

Digital Teaching Intiative Office Hours

Website: https://web.northeastern.edu/nulab/teaching/digital-teaching-integration/

Cara Messina | messina.c@husky.neu.edu | Tuesdays: 1pm - 3pm |
401 Nightingale Hall

Taylor Braswell | braswell.t@husky.neu.edu | Thursdays: 1pm - 3pm |
964 Renaissance Park

Introduction

Module URL: https://notebooks.azure.com/lknelson/projects/introR

R is an incredibly powerful *scripting* language for doing statistical analysis. It's like *Stata* in many ways. Like Stata, it's a logical language, as opposed to a natural language. Natural languages, such as English, carry with them ambiguities: e.g., *Fruit flies like a banana*. Logical languages can not contain any ambiguities. They have a strict and limited *vocabulary* and *syntax*. Like Stata, when writing R code you use the allowed vocabulary and proper syntax to write lines of instructions for the computer to carry out. If you don't use the allowed vocabulary, or the proper syntax, the computer will not understand your instruction and it will give you an **Errror** message. When you get an error message don't panic! Instead, treat it as the beginning of a puzzle, where you have to search through the clues (in the error message) to figure out where you deviated from the allowed vocab and syntax.

Unlike Stata, R is *open source*. It's free, and developers are free to modify and release code based on the R language for use by the community. This is one of the reasons why it is becoming the preferred language for statistics in many disciplines.

Should you choose R or Stata when doing statistics? Stata is a fantastic language. Its learning curve is less steep then R, and its well-controlled releases mean you can trust the output in ways you cannot always do with R. However, the use-cases with Stata can be limited in today's scientific research. With advances in computing power and access to new sources of data, the social science community is rapidly developing new methods beyond traditional econometrics, including machine learning, vector space analyses, and more. Stata does not always incorporate these new methods quickly, while R typically does (sometimes too quickly!). R is also closer to a programming language than Stata, and its flexibility means you can do much more with it, including implementing these powerful new methods, creating interactive visualizations, and even producing apps! While the learning curve is steeper, many think it's worth it.

A quick note to acknowledge that we will be covering A LOT of material today. Your primary goal should be to understand the logic of R and a high-level understanding of how R code runs. Some of the details may move past too quickly. That's OK. You'll have this document to reference as you come back to R on your own. Some extra resources for you and additional opportunities to learn:

- At the bottom of this document there is a *Glossary* of operators and functions. If you forget what a function does, refer to the glossary.
- The DTI will also be holding office hours throughout the semester if you get stuck at any time (see top of page for times).
- We will come back to class in November to check in with your progress.

In sum, embrace this opportunity to learn with an open mind, and forgive yourself (and others) when things get frustrating. No one would expect you to run a marathon the first time you try running. The same, of course, goes for coding.

With that, let's jump in!

Learning Goals

- Understand what R is, why it is useful, and the basics of how to use R for data analysis
- Understand how R interacts with and represents data
- Explore how to use R to manipulate, analyze, and visualize data to get the information you desire

Learning Objectives

- Be able to explain R basics: objects, object types, manipulating objects with operations and functions, and data frames
- Explain how R represents data, and how that allows you to analyze data
- Write enough R code to:
- Read in a data set
- Manipulate and subset variables
- Produce basic summary statistics from a data frame
- Produce (cross) frequency and proportional tables from categorical variables
- Produce two distribution visualizations from a data frame: histograms and boxplots

Lesson Outline

- R basics
- objects
- object types
- object attributes
- manipulating objects
  - operations
  - functions
- Data frames
- reading in data
- slicing data
- Descriptive Statistics
- `summary()`
- `mean()`
- `median()`
- 90/10 ratios
  - `quartile()`
- Producing frequency and proportional tables
- `table()`
- `prop.table()`
- Examining subgroups within a variable using a conditional
- `ifelse()`
- Visualizing data: Histograms and Boxplots
- `hist()`
- `boxplot()`

R Basics

R is oriented around creating *objects*, and then *manipulating* those objects. Most of what you create in R are objects, and almost everything you do in R is manipulating objects.

Each object has a *type*. What we can do to an object depends on its type.

Important object types in R:

- character
- numeric (real or decimal)
- integer
- logical (TRUE and FALSE)
- complex (complex numbers, real and imaginary parts)

We manipulate objects using *operations* and *functions*.

Creating objects

We create, or *assign* objects using the assignment operator `<-`

You can think if it as an arrow pointing to the left. Linguistically, think "I am putting what's on the right, into an object (think: box, or container) on the left."

```
#Note: This is a coding cell.
#The above cell, with the text, is a Markdown cell.

#any text following a hash (#) is a comment
#comments should render in green on your screen
#this text is for human readers: the computer ignores these lines

#To run code in this cell click the >|RUN button at the top
#or press CTL-Enter

#first line of code: assign an object, using <-
out <- 10

#find out the object type
class(out)

#objects can be numerical, and also character
mystring <- "Hello, world"
class(mystring)
```

Manipulating objects

Once we define objects, we manipulate them.

There are two main ways to manipulate objects: *operations* and *functions*.

Important operators in R:

`<-` assignment
`+` addition
`-` subtraction

* multiplication
/ division
^ exponent
%% modulus (remainder from division)
%/% integer division

Relational operators (returns TRUE/FALSE):

> greater than
>= greater than or equal to
< less than
<= less than or equal to
== equal to
!= not equal to

Let's practice!

```
5 + 200

#assign it to an object
mysum <- 5 + 200
mysum

class(mysum)

# other operators
5 * 200
200 / 5

mydiv = 200 / 5

#we can use operators on two objects!
mysum * mydiv
```

What is with the logical, or Boolean, object type (TRUE / FALSE)? These are created through relational operators:

```
4 < 5
mybool <- 4<5
mybool
class(mybool)

#Equivalence (note the double equal)
4==7
```

Collections of obejcts

R also has ways to store collections of objects - typically what we need for statistical analysis.

One of those ways is a *vector*. Note the `c` function: it stands for concatenate. (We'll talk functions in detail in a bit.)

```
myvector <- c(1, 3, 5, 7)
class(myvector)
```

```
#Check if it's a vector (this type of vector is called an atomic vector)
is.atomic(myvector)
```

Functions

We can apply *functions* to objects (we've been doing it all along). Functions are another way to manipulate objects in R.

The syntax here is typically the function name followed by open and closed parentheses: `function_name()`. You *pass* the object you are manipulating into the function by putting it inside the parentheses, and the function returns a value. We can store the returned value in another object. We can also pass a function into a function the same way, which we'll briefly explore below.

We've already been using the functions `class()` and `c()`.

Here are some more. Here, we're passing the object `myvector` into each function and returning a value:

```
length(myvector)
max(myvector)
min(myvector)

#assign the returned value to a new object
myvector_len <- length(myvector)
myvector_len

class(myvector_len)
```

In words, how would you describe what those functions are doing?

R Basics: Recap

To summarize: the R language is organized around creating objects, each of which has a type (e.g., character, numeric, logical). We manipulate those objects using operators (`+`, `-`, `*`, `<`, `>`, `==`), and functions (`function_name(myobject)`).

That's it! That's the basics.

But we can do so much more. First, let's practice what we learned.

```
#Exercise 1:
#In this box, creat a new vector with even numbers between 0 and 10.
# What is the mean and median of that vector?
#(I didn't teach you mean and median,
#but try to intuit it from the functions we did learn.)
```

R for Descriptive Statistics: Summary

For descriptive statistics, we will use a data structure called data frames. If you've ever used a spreadsheet program, or a program like Stata (or SPSS or SAS), you've worked with a similar data structure before. Data frames are flat, rectangular data structures consisting of rows and columns.

Using data frames, we will learn how to:

- slice columns, using the `$` symbol
- produce summary statistics using the `summary()` function
- produce quantiles and other percentiles using the `quantile()` function
- produce cross tables using the `table()` and `prop.table()` function
- conditionally subset a variable using the `ifelse()` function

Data Frames

We can create a data frame from scratch, but more likely, we will read a data frame in from a file saved on our computer.

We'll read a .csv (comma separated values file) in as a data frame object, that we'll call `df`.

```
df <- read.csv(file = "data/inequality_data_1976.csv", header=TRUE, sep=",")
df
```

Check what type of data structure we're dealing with.

```
class(df)
```

In addition to a type, data structures in R also have *attributes*. Attributes are labeled values you can attach to any object.

Let's check the attributes, using (intuitively), the `attributes()` function.

```
#Check the attributes
attributes(df)

#extract the names attribute, which corresponds to the columns
# (or variables) of our data frame
names(df)
```

We can examine each of these columns separately using the `$` symbol:

```
#slice the age column
df$age

#what object type is it?
class(df$age)

#We can do the same functions on it as we need to our vector above
mean(df$age)
median(df$age)
max(df$age)
min(df$age)
```

```
#Exercise 2:
# Find the mean, median, max, and min of a different variable in the data frame.
```

Descriptive Statistics

We typically want to summarize a data frame, which we can do using the `summary()` function.

We can do this on the entire data frame, or one column at a time.

```r
summary(df)
summary(df$age)
```

```r
#We can also pull out specific percentiles, using the quantile() function
#and passing in the percentile we want
quantile(df$age, c(.10))
quantile(df$age, c(.90))

#Reminder: We can assign these to new objects!
age_10 <- quantile(df$age, c(.10))
age_90 <- quantile(df$age, c(.90))

age_10
age_90
```

```r
#Exercise 3 (this one's a challenge!)
# Calculate the 90:10 ratio from the age column.
# If you have time, do the same for a different column.
```

Frequency Tables

We also often want to create frequency tables. We can do so using the `table()` function for raw frequencies, and the `prop.table()` function for normalized frequencies.

```r
#raw frequencies
table(df$sex)
table(df$empstat)

#we can combine to do cross tabluations
table(df$empstat, df$sex)

#proportional frequences
prop.table(table(df$sex))
prop.table(table(df$empstat))

#combined
prop.table(table(df$empstat, df$sex))
```

```r
#Exercise 4:
# Produce a normalized frequency table for two columns of your choice.
# Reminder: what type of columns can be used in frequency and normalized frequency tables?
```

Using a condtion to examine subsets of variables

We can use logical operators to create new variables, or subset our existing variables.

We'll create a variable that indicates whether the respondent is working or not.

```r
#remind ourselves of our data.frame
df
```

```r
#remind ourselves of the values in the empstat column
summary(df$empstat)
```

Now we can create a new Boolean variable: *working*

```r
#check if value is equivalent to 'At work'
df$empstat == 'At work'

#can use Boolean logic for this as well (& = AND, | = OR)

(df$empstat == 'At work') & (df$sex == 'Female')
```

To create a variable contingent of the empstat column, we'll use a new function: `ifelse()`

The syntax is:

```
ifelse(logical operator, value if TRUE, value if FALSE)
```

```r
#create a new variable (or column) based on the equivalency operator

df$working <- ifelse(df$empstat=='At work', 1, 0)
df$working
```

Now we can produce a more succinct contingency table.

```r
prop.table(table(df$working, df$sex))
```

We can also do this without creating a new variable, and simply sub-setting our variable of interest in the `table()` function itself.

Reminder of the syntax:

```
ifelse(condition, value if condition is met, value if condition is not met)
```

The code below is starting to get gnarly, but notice that we're simply stringing functions together, in order to give the computer more sophisticated and precise instructions to get exactly the information we want. In this case, we're stringing together three functions:

- `prop.table()`
- `table()`
- `ifelse()`

Putting these together, we get:

```
prop.table(table(ifelse()))
```

Stringing together functions and operations are the building blocks of R, and is one of the ways R becomes a powerful and flexible language.

```r
prop.table(table(ifelse(df$empstat=='At work', 1, 0), df$sex))

#our values do not have to be numeric
prop.table(table(ifelse(df$empstat=='At work', "Working", "Not working"), df$sex))
```

R for Descriptive Statistics: Recap

For descriptive statistics, we used a data structure called data frames. With this, we:

- sliced columns, using the `$` symbol
- produced summary statistics using the `summary()` function
- produced quantiles using the `quantile()` function
- produced cross tables using the `table()` and `prop.table()` function
- conditionally examined a variable using the `ifelse()` function

The final thing we'll cover is...

Data Visualization!

We'll cover just two here, which can help you visualize the full distribution of your data: histograms and boxplots.

The functions are simple:

- `hist(vector for visualization)`
- `boxplot(vector for visualization)`

```
#histogram of hhincome variable
hist(df$hhincome)
```

```
#box and whiskers plot from hhincome variable
boxplot(df$hhincome)
```

What do you notice about the distribution?

We can set the width and height to make graphs more visually appealing:

```
options(repr.plot.width=5, repr.plot.height=6)
boxplot(df$hhincome)
```

Sandbox time!

With any remaining time, have some fun! I encourage you to create a new .ipynb file to practice creating your own project.

From the home screen on your instance of Azure, click the drop-down menu next to the `+` button. Click "Notebook", and then mark the "R" bubble. You're ready to go!

In your new Notebook, explore our data frame using descriptive statistics, frequency tables, normalized frequency tables, or our two visualizations.

Note the file structure of your project. The .csv file we read it was in the `data` folder. You need to preserve this file structure if you want to upload and read your own data. If you ever get a `Error: File not found` message when trying to read in data, it means you have not pointed R to the correct location for your file.

I recommend uploading any of your own data to the `data` folder, and then keep all of your main `.ipynb` scripts in the same place they are currently located.

Try it out, and have fun!

Glossary of Operators and Functions (selected)

Mathematical operators

`<-` assignment
`+` addition
`-` subtraction
`*` multiplication
`/` division
`^` exponent
`%%` modulus (remainder from division)
`%/%` integer division

Relational operators (returns TRUE/FALSE)

`>` greater than
`>=` greater than or equal to
`<` less than
`<=` less than or equal to
`==` equal to
`!=` not equal to

Logical operators (for Boolean operations)

`!` logical NOT
`&` element-wise logical AND
`&&` logical AND
`|` element-wise logical OR
`||` logical OR

Functions

`attributes()`: names and the dimensions of matrices and arrays; attributes are labeled values you can attach to any object

`boxplot()`: takes in any number of numeric vectors, drawing a boxplot (and whisker plot) for each vector

`c()`: a generic concatenation function which combines its arguments

`class()`: returns the class of an object, or its "internal" type

`hist()`: takes in any number of numeric vectors, drawing a histogram for each vector

`ifelse()`: returns a value with the same shape as the `object` passed in, filled with elements selected from either `yes` or `no` depending on whether the element of `object` is `TRUE` or `FALSE`

`length()`: returns or sets the length of a vector (including a list)

`max()`: returns the maximum of all values in a vector

`mean()`: returns the mean of all values in a vector

`median()`: returns the median of all values in a vector

`min()`: returns the minimum of all values in a vector

`prop.table()`: returns tables of proportions on categorical variables

`quantile()`: measures of position such as quantiles, deciles, and percentiles, created by passing in a vector, matrix, or data frame and the location to measure

`read.csv()`: reads a file in csv format (comma separated values, or any other delimiter separated values format) and creates an R data frame from it

`summary()`: when used on a data frame object, this function shows you a set of descriptive statistics for every variable, depending on variable type

`table()`: returns frequency tables on categorical variables