Introduction to R

aWhere Team

December 15, 2016

What is R?

R is a programming language that is useful for data analysis and visualization. R is an open-source tool, allowing users from across the planet to improve upon R's functionality by fixing bugs in code, and writing packages with new and useful functions, for example.

If you are familiar with Stata, SAS or Python, then the concept of working with a programming language to do data analysis will be familiar to you. For those more familiar with Excel, R is a much more powerful version of the Excel tool. Like in Excel, R can be used for storing data in data frames, similar to an Excel workbook, and can manipulate data using functions such as addition, subtraction, mean, sums, and many more. R is much more powerful than Excel in that it can hold a much larger quantity of data, and provides more advanced analytical functionalities.

Introduction to R resources

This document provides some basic R information. For additional information on R, we recommend free courses including:

1. DataCamp's Introduction to R: https://www.datacamp.com/courses/free-introduction-to-r
2. Swirl courses: https://github.com/swirldev/swirl_courses

Another useful resource, once you're more familiar with R, is the variety of cheat sheets produced by R Studio: https://www.rstudio.com/resources/cheatsheets/

How do I install R?

To begin using R, you must first install R on your computer. For users in Uganda, we recommend installing here: https://cran.usthb.dz/

Next, you'll want an interface to easily interact with R (similar to those in Stata, SAS or Excel). For this, we recommend downloading R Studio:
https://www.rstudio.com/products/rstudio/download3/

Once you've completed your downloads, launch R Studio.

What next? Set up your workspace

Working Directory

Once you're in R, you should understand and set up your environment. The primary thing you want to do is to understand your working directory. This is the location on your computer's hard-drive where documents will be imported from, or exported to, while you do your analysis.

To find your current working directory:

```
getwd()
```

To change or set your working directory, modify the below code to link to the correct file path on your computer:

```
setwd("C:/Users/.../Documents/R/First Analysis")
```

Note that the file paths in R require forward slashes "/" rather than backslashes.

Workspace

Next, you want to understand the R Studio workspace. This cheat sheet is a useful, albeit comprehensive, guide to navigating the R Studio workspace:
https://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf

Broadly, you'll note:

1. A list of open files across the top of the screen.
2. Information about objects (data, etc.) in your "environment"" on the upper right.
3. An information pane on your lower right. This includes tabs for "files", "plots", "packages", "help", and "viewer". This pane provides documentation when you search for it.
4. The console at the bottom of your screen. Here you can actively write code and see results. In general, this should only be used when experimenting, and all other code should be written in your main R file.
5. The main R file in the center of your screen. This is where you'll review and write code.

Documentation

If you need to review documentation files for any reason, type ? and the name of the function you're trying to review in the console.

```
?data.frame
```

Packages

A key feature of R, given its open-source nature, is packages. Packages are written by the R community, and include functions that make it easier to complete certain tasks in R.

Functions can be thought of as simple procedures, such as calculating the mean of a set of data, to more complex procedures, such as running regressions and creating visuals of data.

To install a package, type:

```
install.packages("package_name")
```

There are several packages that are useful for conducting data analysis in R. Some of these include:

1. dplyr
2. data.table
3. ggplot2

We recommend practicing by installing these packages.

```
install.packages("dplyr")
install.packages("data.table")
install.packages("ggplot2")
```

It's useful to note that many packages were already included when you installed R. These are called "base" packages, as they help R run its basic functions. As you become more proficient in R, you will likely come across additional packages that you would like to install yourself. The sky is the limit.

Assignment Operator

In R, objects are assigned a value through an assignment operator. For example, you want to create an object titled "o" with the value of 10:

```
o <- 10
o

## [1] 10
```

The "<-" is the most commonly used assignment operator in R. It can be used to name or assign entire datasets, functions, constants, etc.

```
mean_test <- mean(c(1, 3, 5, 7, 9, 11))
mean_test

## [1] 6

df_test <- data.frame(1:10, 2:11, 3:12)
df_test

##    X1.10 X2.11 X3.12
## 1      1     2     3
## 2      2     3     4
## 3      3     4     5
## 4      4     5     6
## 5      5     6     7
```

```
## 6      6     7     8
## 7      7     8     9
## 8      8     9    10
## 9      9    10    11
## 10    10    11    12
```

Common Object & Data Types

Vectors

A vector is a single number or value (e.g. 1 or "Denver"), or a string of those values (e.g. 1, 2, 3 or "Denver", "Memphis", "New York"). Here are examples of what a vector in R looks like:

```
vector_1 <- c(1:10)
vector_1

##  [1]  1  2  3  4  5  6  7  8  9 10

is.vector(vector_1)

## [1] TRUE

vector_2 <- c("Denver", "Memphis", "New York", "Kampala")
vector_2

## [1] "Denver"   "Memphis"  "New York" "Kampala"

is.vector(vector_2)

## [1] TRUE

vector_3 <- "blue"
vector_3

## [1] "blue"

is.vector(vector_3)

## [1] TRUE

vector_4 <- 1240
vector_4

## [1] 1240

is.vector(vector_4)

## [1] TRUE
```

All of the above are considered vectors. This is confirmed by the "is.vector()" function, which indicates that TRUE, these are vectors.

Lists

Lists are a collection of various R objects, assigned under one name. This can include several single-vector objects (e.g. 1), several multi-vector objects (e.g. 1, 2, 3), data frames, or a collection of different types of objects. Here are examples of what a list in R looks like:

```
list_1 <- list(c(1:10), "blue", data.frame(1:10, 2:11, 3:12))
list_1
```

```
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## [[2]]
## [1] "blue"
##
## [[3]]
##    X1.10 X2.11 X3.12
## 1      1     2     3
## 2      2     3     4
## 3      3     4     5
## 4      4     5     6
## 5      5     6     7
## 6      6     7     8
## 7      7     8     9
## 8      8     9    10
## 9      9    10    11
## 10    10    11    12
```

```
class(list_1)
```

```
## [1] "list"
```

```
list_2 <- list("blue", "red", "green")
list_2
```

```
## [[1]]
## [1] "blue"
##
## [[2]]
## [1] "red"
##
## [[3]]
## [1] "green"
```

```
class(list_2)
```

```
## [1] "list"
```

Notice in list_2 how the format is different from vector_2 above. This helps you distinguish between a list and a vector. You can also confirm that this is a list with the "class()" function, which confirms that both of the above are lists.

Data Frames

Data Frames are the standard data set you've likely worked with in the past. This is a group of rows and columns that contain variables and observations. Data frames can have any number of rows or columns, and the data can contain numeric figures, characters, etc. Here are some examples of a data frame:

```
df_1 <- data.frame(col_1 = 1:10, col_2 = 2:11, col_3 = 3:12)
df_1
```

```
##    col_1 col_2 col_3
## 1      1     2     3
## 2      2     3     4
## 3      3     4     5
## 4      4     5     6
## 5      5     6     7
## 6      6     7     8
## 7      7     8     9
## 8      8     9    10
## 9      9    10    11
## 10    10    11    12
```

```
is.data.frame(df_1)
```

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

```
df_2 <- data.frame(id = 1:10, city = c("Denver", "Memphis", "NA", "New York",
"Kampala", "Unknown", "Tunis", "NA", "London", "Mumbai"))
df_2
```

```
##    id     city
## 1   1   Denver
## 2   2  Memphis
## 3   3       NA
## 4   4 New York
## 5   5  Kampala
## 6   6  Unknown
## 7   7    Tunis
## 8   8       NA
## 9   9   London
## 10 10   Mumbai
```

```
is.data.frame(df_2)
```

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

Data Types

There are several types of data you may work with and encounter in R. You have likely worked with these before even without realizing their names. Common examples of these include:

1. Numeric
2. String/character
3. Logical

Here is an example of a numeric:

```
num <- 1234
is.numeric(num)

## [1] TRUE
```

Here is an example of a string or character:

```
char <- "Denver"
is.character(char)

## [1] TRUE
```

Here is an example of a logical:

```
log <- TRUE
is.logical(log)

## [1] TRUE
```

There are other data types, but these are some of the most common.

Calling APIs, Databases and Other Sources

R is very powerful in that it allows you not only to do data analysis and visualization, but also to retrieve data from external sources. This includes, for example, aWhere's API. API calls in R can be a bit complicated, which is why aWhere has created the aWhere API package in R to simplify. Behind the scenes, the functions in this package call aWhere's API from within R, and return data to R where it can be analyzed.

R allows users to pull or import data from:

1. SQL databases
2. APIs
3. URLs
4. Excel files
5. CSV or Tab files
6. SAS and Stata files

There are other types of data that R is able to use, but this covers some of the basics.

Now what?

You're ready to play with R. Let us know if you have any questions!