Introduction to R for Stata Users

01: Introduction

Purushottam Mohanty

03 June 2021

Why switch to R?

While this is not an exhaustive list, the primary reasons can be listed as:

- R is a free and open source software making it accessible for students and researchers across the world. (goodbye to license worries!)
- R is way faster and resource efficient.
- Syntax much closer to other general programming languages such as Python and can be efficiently scaled up.
- Allows users to take advantage of thousands of packages developed and maintained by a large base of users.
- R is widely used outside academia, thus opening up industry based opportunities in data science and related fields.
- Strong community of of developers and enthusiasts across multiple platforms. (quick stackoverflow solutions!)
- R Markdown allows writing and exporting latex documents, beamer presentations and publish them as HTML or PDFs within R.

Installing R and RStudio

The latest version of **R** can be installed from The Comprehensive R Archive Network (CRAN).

1. Download R for macOS.[1]
2. Download R for Windows.
3. Download R for Linux.

In order to take complete advantage of R, download **RStudio** too. RStudio is a GUI for R that provides code editor, data and graphics viewer, git integration and command line access along with various other features.

1. Download RStudio.[2]

[1] **Note**: Download the Intel installer even if you have a M1 Mac as many libraries haven't been updated to use the ARM version.

[2] **Note**: Download the RStudio Desktop free version. Available for macOS, Windows and Linux.

Basics of R

1. R is an object oriented language. Every dataset, figure, text, string, number or regression result is or can be stored as an object.

2. Depending on what is stored in an object, the `class` of an object is defined which correspondingly determines the operations can be made on the object. (eg: cannot calculate mean of a string)

3. Each object is given a namespace using which we store or call an object. (eg: `x = 1` where `x` is an object.) Namespaces have to be unique or they are overwritten.

4. All operations are performed using functions which can be user-written or accessed from a library.

5. R has different libraries (i.e. packages) which enable additional functions and features beyond the `baseR` library.

Key Differences from Stata

1. In Stata `ssc install` is run once to install and access a particular package whereas in R packages can to be installed once but loaded into memory everytime.

2. In R one can use multiple dataframes (i.e. datasets) simultaneously so no need to constantly dabble with `preserve`, `restore` and `tempsave`.

3. Since, everything is stored as objects in R one easily access elements within objects through consistent syntax.

4. While `putdocx` in Stata can be helpful, RMarkdown provides much easier ways to write latex documents or beamer presentations and export them as PDFs or html.

5. R and RStudio are free and updated regularly so no need to be stuck with older versions (hello Stata 14 users!).

Additional Resources

1. R for Data Science - by Hadley Wickhman and Garrett Grolemund.
2. R Graphics: Codebook - by Winston Chang
3. Introduction to Data Science - by Rafael A. Irizarry
4. Data Science for Economists - by Grant McDermott
5. GSU Library Course
6. R Markdown: The Definitive Guide - by Yihui Xie, J. J. Allaire, Garrett Grolemund

Installing Packages

First, open RStudio > New File > R Script and save it in the directory of choice. Let's install the `tidyverse` and `gapminder` packages using,

```r
install.packages('tidyverse') # composite of data cleaning and visualization packages
install.packages('gapminder') # socio-economic data for countries
```

After installation, the packages need to loaded into memory. Note that packages which have been installed earlier need not be installed again and can directly be loaded into memory.

```r
library(tidyverse)
library(gapminder)
```

A more effective way to do the installation and loading process is using the `pacman` package. Just install the package once using `install.packages('pacman')` and then use,

```r
pacman::p_load(tidyverse, gapminder)
```

The `PACKAGE::function()` syntax allows one to use functions from installed packages without loading them.

Object Types

R objects can be of different `class` depending on the type of data that is assigned to them.

```r
# character
x = "London"
# numeric
x = 100
# list
x = as.list(c("London","Paris","Zurich","Madrid","Amsterdam"))
# dataframe
x = read_csv("./example.csv")
```

There are a large number of object types used in R but `character`, `numeric`, `list`, `dataframe` and `matrix` are the most commonly used ones. Objects can also be transformed between types.

The object type determines the operations that can be performed on the object.

```r
x = 1
y = "A"
x + y # cannot be performed (numeric + character)
```

Reading Files

Before we proceed to reading different file formats, let's set the base directory first,

```r
getwd() # getting current directory
setwd('/Users/purushottam/Documents/GitHub/prog-notes/stataR') # setting directory
```

```r
# comma delimited file
library(readr)
read_delim(file = './example.csv', delim = ',')
# excel file (reads .xls and .xlsx files)
library(readxl)
read_excel(path = './example.xlsx', sheet = 'sheet1')
# stata file
library(haven)
read_stata(file = './example.dta')
# json file
library(jsonlite)
read_json(path = './example.json')
```

Similarly, there are other packages to load SAS, SPSS, TIFF or other files.

Dataframes

Datasets in R are called `dataframe`. If you tried any of the functions listed earlier to read files you would notice that while they appear in the console, there is no way to access them. In order to do so, we need to store them as an object. This can be done using,

```
df ← read_stata(file = './example.dta')
```

The .dta file is now stored as a `dataframe` class object `df` and can be viewed either by selecting it in the environment window in R Studio or by simply executing `df`. Note that many R users like to use `=` instead of `←` as the assign function. They both work the same so pick what you want but stay consistent.

Datasets can either be imported into R as dataframe or it can be created in the following way.

```
df = data.frame(names = c("London","Paris","Zurich","Madrid","Amsterdam"),
                rents = c(1800, 1400, 2000, 1000, 1700))
```

Dataframes (cont.)

For the current tutorial however we shall data from `gapminder` package. The package provides life expectancy, GDP per capita, and population data at the country level for every 5 years from 1952 to 2007.

```
df = gapminder
df
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
##  6 Afghanistan Asia       1977    38.4 14880372      786.
##  7 Afghanistan Asia       1982    39.9 12881816      978.
##  8 Afghanistan Asia       1987    40.8 13867957      852.
##  9 Afghanistan Asia       1992    41.7 16317921      649.
## 10 Afghanistan Asia       1997    41.8 22227415      635.
## # … with 1,694 more rows
```

Indexing

All elements within objects in R can be accessed using row and column index or names. R follows a consistent form of indexing which is applicable to all class of objects.

A `dataframe` in R can be indexed by using the following syntax `dataframeName[rowNumber, columnNumber]` or `dataframeName[rowNumber, columnName]`.

```
df[1,4]
```

```
## # A tibble: 1 x 1
##   lifeExp
##     <dbl>
## 1    28.8
```

```
df[1, 'lifeExp']
```

```
## # A tibble: 1 x 1
##   lifeExp
##     <dbl>
## 1    28.8
```

Indexing (cont.)

Multiple rows or columns can also be indexed.

```
df[1, 4:5] # row 1 and column 4-5
```

```
## # A tibble: 1 x 2
##   lifeExp     pop
##     <dbl>   <int>
## 1    28.8 8425333
```

```
df[1, c('lifeExp','pop')]
```

```
## # A tibble: 1 x 2
##   lifeExp     pop
##     <dbl>   <int>
## 1    28.8 8425333
```

Indexing can also be done for only rows or columns.

```
df[1,] # row 1 - all columns
df[,4] # all rows - column 4
df[4] # all rows - column 4
```