AfCore.dll

Introduction to RStudio Environment

Basic Features

Compiled by Haymanot Berelie

May 25, 2023

Assosa, Ethiopia

OVERVIEW OF RSTUDIO

Why Rstudio?

Introduction to RStudio

⟾RStudio is a powerful and productive user interface for R or Integrated Development Environment (IDE) for R and makes using R easier.

⟾Organize your code, output, and plots.

⟾Auto-complete code and highlight syntax.

⟾Help view data and objects.

⟾Enable easy integration of R code into documents.

⟾User-friendly interfaces.

⟾ Open source, but paid option as well (not worth it for typical users)

⟾ Same install process

Benefits of RStudio

⟹ Easier transition from Stata to R

⟹ Multiple window views at once – Results, Code, Working Environment, and Graphs

⟹ If something on your computer crashes while you're working, R will have your code waiting for you when you re-open RStudio.

⟹ Auto suggesting the function And re-type everything all over again when a mistake make in typing code into the console.

Getting started with RStudio

⟹ **Once you have installed RStudio, there will be an icon on your desktop.**

⟹ **Double click it and RStudio will start up.**

⟹ **Start the R system, the main window (RGui) with a sub window (R Console) will appear. In the `Console' window the cursor is waiting for you to type in some R commands.**

⟹ **R does have a few pull-down menus,**

The menu bar

The menu bar in RStudio contains 11 pull down menus, which are:

1. **File:** The file menu contains options for **new file, new project, opening, saving,** and **printing R documents, importing dataset, compile pdf,** as well as the option for **exiting the session or project (**clicking on the **x** in the R window or typing the command q() on the console).

⟹ New file: open new r script, python scripts ,shell, Markdown, notebooks, html file, latex or sweave document…

⟹ New project: for handling the analysis of new data and keep everything together.

⟹ Open: data, project in the new sessions, script, and other documentation

Cont'd..

⇒ saves everything; command, output and most recommended. Save history
⇒ saves commands and objects and save image # just saves the objects in the workspace .
⇒ Import dataset: importing external data from xx to RStudio
⇒ **Load Workspace** and **Load History** are options to open previously saved work.

2. **Edit:** The edit menu contains the standard functionality of undo, redo, **cut, copy, paste, ,find and replace, check spelling, word count, Clear output and /or console.**

3. Code: The code menu contains options for insert chunk, jump to, go to file/function, show document outlines/diagnostics/diagnostic project, extract variable/function, comment/uncomment/reindent/run lines, and source.

Cont'd…

4. View: in this drop down menu we can modify the size of panes, taps, positions … etc. The view menu contains options for show/hide panes, zoom, switch to/next/previous/first and last tabs, move focus to xx , show history/files/packages…etc.

5. Plots: The plot menu contains options for show, save, and remove plots

6. Session: The session menu contains options for new and quit sessions, interrupt, terminate, restart R, setwd(), load, save, and clear

Cont'd…

7. Build: The build menu contains options for configure build tools

8. Debug: The debug menu contains options for toggle and clear all breakpoints, execute next line, step into function, finish function continue/stop/helping debugging.

9. profile: The profile menu contains options for start/stop/open/help profiling, profile selected lines.

Cont'd…

10. Tools: The view menu contains options for install and update packages, version control, shell, terminal, jobs, add-in, memory, help and modify keyboard shortcut, project and global options…etc.

11. Helps: The help menu contains options for R help, about RStudio, community forum, and docs, markdown and roxygen quick reference, accessibility, cheat sheets, and diagnostics.

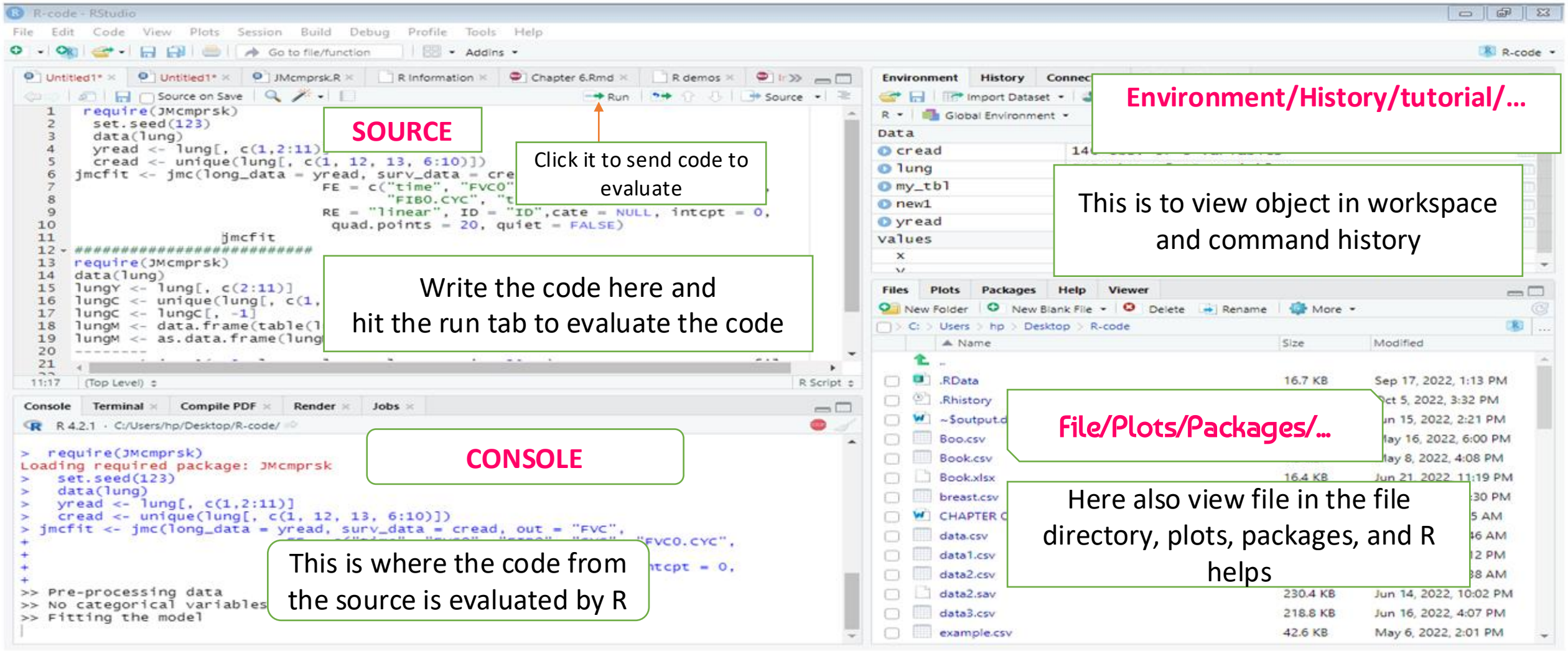
RStudio Layout or panes

- Console
  - Here is where R actually evaluates code and get an immediate response.
  - type the code into the Source, and then send the code to the Console
  - We can type code directly into the console
- Source
  - The source pane is where you create and edit ``R Scripts'' - your collections of code
  - Equivalent of Do file editor in Stata
  - Not open by default

cont’d…

- Environment
  - This pane includes by default environment/history/connection/tutorial tabs.
  - Shows the names of all the data objects
  - shows a history of all the code previously evaluated .
  - clickable actions like “Import Dataset"
- File/Plot/Packages/Viewer/Etc.…
  - Show list, install , update, and remove packages
  - access to the file directory and shows all your plots
  - Help tabs for R functions.

4 primary windows



WORKSPACE AND RSTUDIO PROJECT MANAGEMENT

⟹`getwd()` **# shows the working directory**

⟹`lits.d()` **# lists the working directory**

⟹`setwd()` **# sets the working directory**

⟹`choose.dir()` **# will prompt you to choose your directory (interactive)**

⟹`setwd("D:/RTraining/")` **# this changes the wd Observe the forward slash or use double backslash.**

⟹`Setwd("D:\\RTraining\\")` **# same like above**

⟹`dir()` **# lists files in the working directory**

⟹`list.files()` **# lists files in the working directory**

To Create an R project, go to:

⟹ **File > New Project and then choose: New Directory> Name for the directory > Click on Create Project.**

⟹ **Or in the top right corner press the create project tab > New Project and then choose: New Directory> Name for the directory > Click on Create Project**

⟹ **Or in the top left corner press the project(none) icon > New Project and then choose: New Directory> Name for the directory > Click on Create Project**

⟹ **can also create via this syntax**

```
> install.packages("prodigenr")
```
 # to install this package

```
>Library(prodigenr)
```
 # to call the package

```
> setup_project(" the path of the work directory")
```

For more complex project it may be useful to create sub-directories to contain data, scripts and other documents separately. Can also type the below function into the Console, but we won't do that in this session.

Data permanency and removing objects

- The entities that R creates and manipulates are known as objects. These may be variables, arrays of numbers, character strings, functions, or more general structures built from such components.

- During an R session, objects are created and stored by name (we discuss this process in the next section). The R command
- to display the names of (most of) the objects which are currently stored within R.

  - `objects()`
  - `ls()`
- To remove objects the function rm is available:
  - `rm()`
- To remove all the objects in R type:
  - `rm(list=ls(all=T))`

Data structure and types

Basic data type in R

- Logical
- Numeric
- Integer
- Character
- Complex
- **Raw**

Basic data structures in R

- ✓ Vector
- ✓ Factor
- ✓ List
- ✓ Matrix
- ✓ data frame

examines features of vectors and other objects

functions	purpose
↳`class()`	↳What kinds of object is it(high level)
↳`typeof()`	↳What is the object's data type(low-level)
↳`mode()`	↳What is the object's data type(often the same as `typeof()`)
↳`length()`	↳How long is it?
↳`attribute()`	↳Does it have any metadata
↳`Str()`	↳What is structure of the object?

Cont'd…

Data type can be converted from one to other using as.something() and can be checked using is.something() code.

Vectors and assignment

Assignment can be made using any of the following operators.

1. `a <- 7` # keyboard shortcut for the operator <- "alt + -"
2. `a=7`
3. `assign("a",7)`
4. `7->a`

vectors

⟹**Vectors are the simplest type of object in R. They can easily be created with c, the combined**

function.

⟹**There are 3 main types of vectors:**

- ❑ **Numeric vectors**
- ❑ **Character vectors**
- ❑ **Logical vectors**

Numeric Vector: Is a single entity consisting of an ordered collection of numbers.

Example: `x<-c(10, 6, 3, 6, 22)`

```
x<-c(10, 6, 3, 6, 22)
```

Character and logical vector

⟹**Character strings are entered using either matching double(“ ") or single(' ') quotes, but are printed using double quotes(or sometimes without quotes).**

⟹**To set up a character/string vector z consisting of 3 place names use:**

Example: `> z<-c("Canberra","Sydney","Newcastle")`

Or

```
>z<-c('Canberra','Sydney', 'Newcastle’)
```

Logical Vectors

⟹**A logical vector is a vector whose elements are TRUE, FALSE or NA.**

```
> cat(is.na(z)) or print(is.na(z))
```

[1] FALSE FALSE FALSE TRUE

matrices

- **Matrices a generalization of a vector with the same data type. A matrix can be generated in two ways.**

1 Using the function dim

```
>x <- c(1:8)
> dim(x) <- c(2,4)
```

2 Using the function matrix:

```
>x <- matrix(c(1:8),2,4,byrow=F)
```

- **Use the function cbind(rbind) to create a matrix by binding two or more vectors as column(row) vectors**

arrays

Arrays are generalizations of vectors and matrices

That means, vectors in the mathematical sense are one-dimensional arrays where as matrices are two-dimensional arrays; higher dimensions are also possible.

There are two methods of creating arrays in R

Method 1: Using vectors

A vector can be used by R as an array only if it has a dimension vector as its dim attribute.

Example: array with dimension vector c(3,5,100) and a vector of 1500 elements.

```
> z=c(1:1500)
> dim(z) <- c(3,5,100)
```

Cont'd…

Method 2: Using the function array()

Example: array with dimension vector c(3,5,100) and a vector of 1500 elements.

```
>z=array(1:1500, dim= c(3,5,100))
```

- **Or**

```
>h<-c(1:1500)
>z<- h ; dim(z) <- c(3,5,100)
```

list

Lists: are collections of arbitrary objects. That is, the elements of a list can be objects of any type and structure.

Lists are created with the list() function:

```
>L<-list(object-1,object-2,…,object-m)
```

Example:

```
>L <- list( c(1,5,3), matrix(1:6, nrow=3), c("Hello","world") )
```

- **The elements of the list are accessed with the [[ ]]operator.**
- **Examples: Consider the previous example**

```
> L[[1]]          # First element of L
```

Data frames

Data frames: regarded as an extension to matrices.

Data frames can have columns of different data types and are the most convenient data structure for data analysis in R.

Data frames are lists with the constraint that all elements are vectors of the same length.

The command data.frame() creates a data frame:

```
>dat<-data.frame(object-1,object-2,…,object-m)
```

Example:

```
> name= c("Eden","Solomon","Zelalem","Kidist")
> age=c(18,22,25,27)
> sex=c("F","M","M","F")
```

Cont'd…

```
> stud=data.frame(name,age,sex)
> Stud
```

- **To display the column names:**

```
> names(stud)      #OR  colnames(stud)
```

- **To display the row names:**

```
>rownames(stud)
```

- **to change the column names:**

```
>names(stud)<-c("name","age","sex")
```

- **to change the row names:**

```
>row.names(stud)<-("Wrt","Ato","Ato2","Wro")
```

factors

- **Factors provide compact ways to handle categorical data**
- **An R factor might be viewed simply as a vector with a bit more information added. That extra information consists of a record of the distinct values in that vector, calledlevels. Here's an example:**

```
> x <- c(5,12,13,12)
```

```
> xf <- factor(x)
```
 # The distinct values in xf—5, 12, and 13—are the levels here.

[1]5 12 13 12

Levels: 5 12 13

Common Functions Used with Factors

```
tapply(): tapply(x, xff, mean)
```

```
split()
```
 # , which splits a vector into groups and then applies a specified function on each group

```
by()
```

Reading(importing) data

Large data objects will usually be read as values from external files rather than entered during an R session at

the keyboard. In R you can import text files with the function read.table

Syntax: read.table(“file name", arguments)

Example:

```
> getwd()
[1] "C:/Users/hp/Desktop/R-code"
> myfile<- "C:/Users/hp/Desktop/R-code/Data.txt"
> mydf <-read.table(myfile, skip=3,sep=",",header=TRUE)
```

other functions

⇒`read.csv` **# to read —comma separated values|| files period as a decimal**

⇒ `read.delim` **# to read- tab delimited files period as a decimal**

⇒`read.mtp` **is functions to read Minitab files**

⇒`read.spss` **# to reads a file stored by the SPSS save or export commands.**

⇒`read.csv2` **# to reads a file separated by semicolon (;) and decimal by comma (,)**

⇒`read.delim2` **# to reads file separated by tab (\t) and decimal by period (.)**

⇒**To see the built-in dataset currently available use the following command:**

```
>data()
```

Writing(export) data

The data (usually a matrix) x are written to file file. If x is a two-dimensional matrix you need to transpose it to get the columns in file the same as those in the internal representation.

 # To write a CSV file for input to Excel one might use

```
>write.csv(,file = "filename.csv")
```

```
>write.table(x, file = "foo.csv", sep = ",", col.names = NA, qmethod = "double")
```

 # Writes a matrix or data frame to a file or the console, using column labels and a layout respecting columns.

```
>write.matrix(x, file = "", sep = " ", blocksize)
```

To save your output from a session in R can be saved using the sink command:

```
> sink("Rintro")
```
 # if you want to save output as word use .doc/docx file extensions

To turn off the sink command:

```
> sink()
```

Many thanks!!

Haymanot Berelie