FDP – R for Data Science

Session 1: Coding syntax and using the built-in functions

Prerequisite

Download R - https://cran.r-project.org/bin/windows/base/
Download RStudio - https://rstudio.com/products/rstudio/download/
Material and Files - https://github.com/DG1606/FDP2020

R

R is a statistical programming language and is a wonderful tool for statistical analysis, visualisation and reporting. It is a free open-source software. Comprehensive R Archive Network (CRAN) is the maintainer of R.
Version: 4.0.0 (24-04-2020) Nickname – Arbor Day

RStudio

RStudio is an integrated development environment (IDE) that consolidates basic developer tools for R into a single, user-friendly interface. RStudio makes writing and executing R code as effective and easy as possible.

Open RStudio.
In menu, click *File –> New File –> R Script*.

The RStudio interface now consists of four main panes, or windows in the below order(default):

- **Bottom left**: console or command window. Here you can type any valid R command after the > prompt followed by Enter and R will execute that command.
- **Top left**: text editor or script window. This is where you can save and edit collections of commands.
- **Top right**: environment & history window. The environment window contains objects (data, values, functions) R has currently stored in its memory. The history window shows all commands that were executed in the console.

The environment window gives you an overview of your current workspace. The collection of objects currently stored is called the workspace.

- **Bottom right**: files, plots, packages, help, & viewer pane. Here you can open files, view plots, install and load packages, read man pages, and view markdown and other documents in the viewer tab.

You can change the pane layout. Go to *Tools –> Global Options –> Pane Layout*
You can collapse a pane that you don't find useful. You can resize the size of each pane by simple dragging,

Tip: CTRL+1 and CTRL+2 will alternate between script and console.
Hands-on: Explore the RSudio IDE Cheat Sheet for other shortcuts

Coding in script or console? If you enter code directly into the console, it will not be saved by R: it runs and disappears. Instead, by typing your code into a script file, you are creating a reproducible record of your analysis. An R script is just a plain text file that you save R code in. The code you type is called a command.
Tip: The console supports the ability to recall previous commands using the arrow keys.

Setting the working directory: Each time you open R, it links itself to a directory on your computer, which R calls the working directory. Working directory is a folder where R reads and saves files. It helps to keep all your data, scripts, image outputs etc. in a single folder. R is always pointed at a directory on your computer. You can find out which directory by running the getwd() (get working directory) function; this function has no arguments. To change your working directory, use setwd() and specify the path to the desired folder. You can see what files are in your working directory with list.files().

Tip: On a Windows machine, if you copy and paste a filepath from Windows Explorer into RStudio, it will appear with backslashes (\ ), but R requires all filepaths to be written using forward-slashes (/), so you will have to change those manually.
Tip: You can also use the menu to change your working directory under *Session –> Set Working Directory –> Choose Directory* OR you can set a default working directory through Global Options.

R Packages
A package is essentially a library of prewritten code designed to accomplish some task or a collection of tasks. An R package bundles together useful functions, help files, and data sets. Usually the contents of an R package are all related to a single type of task, which the package helps solve.

By default, R installs a set of packages during installation. The set of packages loaded by default –
getOption("defaultPackages")
More than 19,600 user created packages are available for use through CRAN repository as of early May 2020. There are multiple ways to install packages.
install.packages("nameofpackage")
R will download the package into a libraries folder on your computer. After a package has been installed, it needs to be loaded before it can be used.
library(nameofpackage)

Hands-on: Install and load packages dplyr and ggplot2. To see which packages you currently have in your R library, run library().
Tip: You only need to install packages once, so you can type directly in the console box, rather than saving the line in your script and re-installing the package every time. Alternatively, you can use the Packages tab to install additional packages into your RStudio.

How to choose the right R Package?
There are thousands of helpful R packages for one to use, but navigating them all can be a challenge. Moreover, many R packages do the same things. Few of the most essential packages are listed below –

XLConnect, xlsx - These packages help you read and write Microsoft Excel files from R.
dplyr - Essential shortcuts for subsetting, summarizing, rearranging, and joining data sets. dplyr is our go to package for fast data manipulation.
tidyr - Tools for changing the layout of your data sets. Use the gather and spread functions to convert your data into the tidy format, the layout R likes best.
ggplot2 - R's famous package for making beautiful graphics. ggplot2 lets you use the grammar of graphics to build layered, customizable plots.
caret - Tools for training regression and classification models.

How to get Help in R
Before we start writing code, it is important to know how to get help.
Start with Google. Include [R] to your query/search to restrict it to relevant results. As a beginner in R, preferably look for answers from stackoverflow and R-bloggers. Stack Overflow is a website that allows programmers to answer questions and users to rank answers based on helpfulness.
If you want help on a function or a dataset that you know the name of, type ? followed by the name of the function in the console. To find functions, type two question marks ?? followed by a related keyword.
The alternative to ? is help("functionname") and for ?? is help.search("keyword").

Most functions have examples that you can run to get a better idea of how they work.
example("nameofpackage")

Tip: The Help tab allows you to search for help directly from your RStudio window.

Basic coding syntax etiquette & support

Comment:
The best way to split your script into sections is to use comments. You can define a comment by adding hashtag (#) to the start of any line and typing text after it. R will not run anything that follows a # on a line. Use comments to document your code, so that you can remember what you have done and also make it easier for others to follow what the code is doing.

Naming files and objects:
File names for scripts should be meaningful and end in .R. Avoid spaces and funky characters!!! Use an underscore to separate words within a script file.
The preferred form for object/variable names is snake_case where you separate lowercase words with underscore (_).
For functions, all lower-case letters and words separated by dots.
Remember: R is case-sensitive.

Spacing:
Place spaces around all operators (=, +, -, <-, etc.). The same rule applies when using = in function calls. Always put a space after a comma, and never before, just like in normal prose.

Code completion:
RStudio supports automatic completion of code using the Tab key.

Keyboard shortcuts:
Refer Tools in menu.
Example-
Alt+- is Insert assignment operator
Ctrl+Enter is Run selected line(s)

Cheat sheets:
Refer Help in menu.

Accessing built-in datasets

Around 100 datasets are supplied with base R distribution (in the package- datasets), and others are available in add-on packages. To see the list of datasets currently available use data(). All the datasets supplied with R are available directly by name.
mtcars, iris, titanic, flights etc are a few of the very commonly used datasets for practicing and experimenting.

Basics of R

Saving code in an R script

The usual workflow is to save your code in an R script (".R file"). Go to *File –> New File –> R Script* to create a new R script. Click on the "Save" icon to save the file under a desired name in your working directory. The code in your R script can be run by placing the cursor on the concerned line of code, and then either pressing Ctrl+Enter OR by clicking the Run button.

Mathematical operations

Being a statistical programming language, R can be used for mathematical operations.

```
# Mathematical Operations
1 + 1
1 + 2 + 3
2 - 3
2 * 3
2 * 3 + 1
5 / 2
3^2
5 %% 2 # Modulo, gives the remainder
```

Variables

Most of the times we are interested in storing results for reuse. A variable is a name for a value. We can assign a variable using the assignment (<- or =) operator. The arrow form (<-) of assignment is the most common form among the R community.
Remember the keyboard shortcut?
RStudio shows us the variables in the "Environment" pane. We can also print a variable by typing the name of the variable and running the command. Spaces aren't ok inside a variable name. Dot (.) and underscore (_) are ok. Numbers are ok, except as the first character.
All objects (variables, arrays of numbers, character strings, functions etc.) created during an R session can be stored permanently in a file for use in future R sessions. At the end of each R session you are given the opportunity to save all the currently

available objects. If you indicate that you want to do this, the objects are written to a file called .RData in the current directory, and the command lines used in the session are saved to a file called .Rhistory.

```
# Variables
# Assign a value to a variable
x1 <- 2
x2 <- -4
x2
(x3 <- 1:5) # observe the console. Colon operator creates a sequence
x4 <- x5 <- 3
assign("x6", 9)
x7 <- "Hello World"
print(x7)
x8 <- 5.5
x1 <- 3.5
x9 <- 5L #L is used to indicate integer
x10 <- seq(1:5)
x10
my_marks_in_r <- 75
print("I have got a hang of this")
# Remove a variable
rm(x5)
x5 # understand the error

# Use the created variables/objects in calculations
x1 + x2
my_sum <- x8 + x9
print(my_sum)
x9 * 5
```

Data Types

The most common data types are numbers, which R calls numeric values, and text, which R calls character values.

```
# Data types
class(x1)
class(x9)
class(x7)
typeof(x10)
is.numeric(x2) # Alternate way of asking the variable its type
```

```
# Dates
today_date <- as.Date("2020-05-22")
today_date
class(today_date)
today_date + 7

# Logical
is_r_easy <- TRUE
class(is_r_easy)
2 > 3
1 + 1 == 3 # is 1+1 equal to 3?
1/49 * 49 == 1 #Surprised??
```

Built-in Functions

The functions which are already created or defined in the programming framework are known as a built-in function. R has a rich set of functions that can be used to perform almost every task. We will understand a few of them here and learn many more as we progress.

```
# Built-in Functions
# Numeric Functions
abs(-4.5)
sqrt(16)
ceiling(4.565)
floor(4.565)
trunc(4.565) # For positive numbers, trunc and floor give the same result.
ceiling(-4.565)
floor(-4.565)
trunc(-4.565) # This is because trunc always rounds toward 0
round(4.565)
round(4.565, digits = 2)
round(-4.565)
round(-4.1)
signif(4.565, digits = 2)
log(10)
exp(1)
factorial(3)
```

```
# Character Functions
word <- "programmingwithR"
substr(word, start = 12, stop = 15) # extract
substr(word, start = 12, stop = 15) <- "thru" # replace
word
substring(word, first = 12)
substring(word, first = 12, last = 15)
word_1 <- "Split the words in a sentence"
strsplit(word_1, " ")
full_name <- "Dhimant Ganatra"
strsplit(full_name, " ")
my_firstname <- strsplit(full_name, " ")[[1]][1]
my_firstname
rating <- c("A", "B", "A+", "AA", "C")
grep("A", rating) #Globally search a Regular Expression and Print
news <- "The traffic in Bangalore is chaotic. Bangalore is an IT city"
sub("Bangalore", "Bengaluru", news) # replace first match
gsub("Bangalore", "Bengaluru", news) # replace all match
toupper(news)
tolower(news)
```