Basics

Contents

This note will review some of the Basics of R.

Command line Operations

You can use R for simple arithmetic

```
45 * 34
```

```
## [1] 1530
```

```
2^100
```

```
## [1] 1.267651e+30
```

Objects and Assignment

Most analyses do not stop at elementary hand calculator type operations, so it is important to be able to save intermediate computations in an object. Process of assigning values to an object is called Assignment. Once values are assigned to an object, operations can be conducted on the objects.

```
a = 45
b = 34
a * b
```

```
## [1] 1530
```

In the above example, assignment was denoted using `=`. When using `=` for assignment, it is important to place the object on the left. Please note, most published R code uses `<-` for assignment. The use of `=` is my personal preference as it is one less keyboard key to click. When using the `<-`, the direction of the arrow indicates where the object is.

```
a = 45
# 45 = a    Do not do this as it will throw an error
b <- 34
a * b -> c  # When using the arrow, object can be on left or right
c
```

```
## [1] 1530
```

Objects created in R are stored in the environment and can be seen in the Environment pane of RStudio or can be listed by calling `ls()`

```
ls()
```

```
## [1] "a" "b" "c"
```

Or `objects()`

```
objects()
```

```
## [1] "a" "b" "c"
```

In R, an object could be a variable, a dataset, a paragraph of text, or a mathematical index. We can perform operations on objects. Objects are generally created via assignment or imported from an external sources, however there are a few built-in objects like `pi`.

Names

Object names must start with a letter, and can only contain letters, numbers, underscore and period. Aside from that, naming style is a personal choice, but whatever your style, be consistent. Here are a few commonly used styles

```
my_first_variable = 2010
MyFirstVariable = 2010
my.first.variable = 2010
```

Which of the following are acceptable names for an object?
1. Var-name
2. Var.name
3. Var name
4. Var%name

Functions

R has a large collection of built-in functions that have the following syntax:

```
function_name(arg1 = val1, arg2 = val2, ...)
```

Functions generally take one or more arguments. Arguments may be a mix of required and optional arguments. As the name suggests, required arguments must be fed a value. On the other hand, if an optional argument is not provided a value, it will take on a default value. To learn more about a function, enter a `?` followed by function name as shown below.

```
?log
```

It is a good practice to include argument names along with values in the function call. It is also best to enter arguments in the same order as the function definition. As an example, let us explore the function, `log`. If you check the help file by running above code, you will note that `x` is a required argument while `base` is an optional argument with a default value of exp(1).

```
log(x = 100, base = 10)
```

```
## [1] 2
```

Names of functions are generally intuitive. For instance, to compute square root, use `sqrt()`.

```
sqrt(81)
```

```
## [1] 9
```

A web search using say, Google, for the operation followed by `R` is an effective way to find a function. For e.g., a Google search for "cosine in r". But, always check the documentation for a function before using it. As noted above, you can do this by typing a question mark before the function name and running it.

Data Types

There are a number of supported data types which are referred to as a class. Here are a few commonly used data types and corresponding class names in R.

- Numbers: numeric, integer
- Text: character
- Binary: logical
- Complex: complex
- Categorical: factor
- Date and Time: Date

One can find the class of an object using the `class` function

```
class(45)
```

```
## [1] "numeric"
```

```
class(45 > 34)
```

```
## [1] "logical"
```

Class of an object may be changed either automatically to accommodate an operation or manually. As an example of automatic class changes, consider a mathematical operation on an object that is logical.

```
10 + T - 41 * F
```

```
## [1] 11
```

Manual changes to an object class known as coercion can be done through `as.` functions. Consider the following.

```
as.logical(11)
```

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

```
as.character(TRUE)
```

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

Data Structures

A data structure contains a set of objects. There are a number of different data structures in R. A few of the commonly used data structures classified by type of objects and the shape of the structure are

- vector: one-dimensional, all objects belong to same class
- list: laundry-bag type flexible structure that can contain objects of different classes
- matrix: two-dimensional structure, all objects belong to same class
- data.frame: two-dimensional structure, where each column contains objects of the same class

vector

One-dimensional data structure where all objects belong to same class.

```
set_of_integers = c(4,5,2,1)
is.vector(set_of_integers)
```

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

```
consecutive_integers = 4:9
is.vector(consecutive_integers)
```

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

```
names = c('Nikhil','Rohan','Sharmila','Vishal')
is.vector(names)
```

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

Element-wise operations is the default

```
vec = c(1,2,3,4)
vec * vec
```

```
## [1]  1  4  9 16
```

If vectors are of unequal length, elements of shorter vector will be recycled.

```
vec = c(1,2,3,4)
vec * c(1,2)
```

```
## [1] 1 4 3 8
```

Matrix-operations need to be explicitly specified

```
vec %*% vec
```

```
##      [,1]
## [1,]   30
```

Automatic Coercion. When faced with objects of different classes, c() will coerce objects to a common class based on a priority order: Logical –> Numeric; Numeric –> Character; Logical –> Character

```
name_and_age = c('Nikhil', 8, 'Rohan', 10, TRUE, FALSE)
is.vector(name_and_age)
```

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

```
class(name_and_age)
```

```
## [1] "character"
```

matrix

Two- dimensional data structure where all objects belong to same class. Like Vectors, default is element-wise operations.

```
matrix_1 = matrix(1:8,nrow=2,ncol=4)
matrix_2 = matrix(8:1,nrow=2,ncol=4)
matrix_1*matrix_2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    8   18   20   14
## [2,]   14   20   18    8
```

list

Allows elements of different classes to exist.

```
name_and_age = list('Nikhil', 8, "Rohan", 10)
class(name_and_age)
```

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

data.frame

Two-dimensional data structure which resembles a spreadsheet. Elements in a column belong to the same class, but columns can belong to different classes. One can create a dataframe using `data.frame()`

```
set.seed(10)
df = data.frame(id = 1:10,
           gender=sample(c('Male','Female'),size = 10,replace = T),
           attended=sample(c(T,F),size = 10,replace=T),
           score=sample(x = 1:100,size = 10,replace = T), stringsAsFactors = T)
class(df)
```

```
## [1] "data.frame"
```

```
head(df)
```

```
##   id gender attended score
## 1  1   Male     TRUE    95
## 2  2   Male    FALSE     8
## 3  3 Female     TRUE     7
## 4  4 Female     TRUE    27
## 5  5 Female    FALSE    86
## 6  6   Male     TRUE    82
```

However, dataframes are seldom created in this manner. It is more common to read in data from an external source as a dataframe. Text files with comma as a delimiter may be read using `read.csv()`. To read in a file, one must use one of the two methods:

- Method 1: Specify the full path of the file in `read.csv()`, or
- Method 2: Set the working directory to the location of the file. Then specify file name in `read.csv()`

To find the file path:

- For Windows: With Shift pressed, right click on file. Select "Copy as Path"
- For Mac: Right click on file and press Option. Select "Copy xxx as Pathname"

Read in a data.frame

Here is an illustration on how to read in a file called `info.csv` that is saved in the folder, `c:/my_classes/best_course_ever` (or (`/my_classes/best_course_ever` on a Mac) using `read.csv()`

Method 1

```
read.csv('c:/my_classes/best_course_ever/info.csv') # on Windows
read.csv('/my_classes/best_course_ever/info.csv') # on Mac
```

```
##   id     name gender age love_fruits
## 1  1    Rohan   Male  10        TRUE
## 2  2   Nikhil   Male   8       FALSE
## 3  3 Sharmila Female  40       FALSE
## 4  4   Vishal   Male  44        TRUE
```

Method 2

```
setwd('c:/my_classes/best_course_ever/') # on Windows
setwd('/my_classes/best_course_ever/') # on Mac

read.csv('info.csv')
```

```
##   id     name gender age love_fruits
## 1  1    Rohan   Male  10        TRUE
## 2  2   Nikhil   Male   8       FALSE
## 3  3 Sharmila Female  40       FALSE
## 4  4   Vishal   Male  44        TRUE
```

Packages

The R System involves a Base System and a set of Packages. A Package is a collection of code and functions written for the R language Usually focuses on a specific task or problem. Implementation is described in the package documentation available on CRAN. By only installing packages that one needs, R can be kept light. Most useful R applications appear in packages.

To use a package, install it once. Then call the package for each R session. Here is an illustration with the `ggplot2` package.

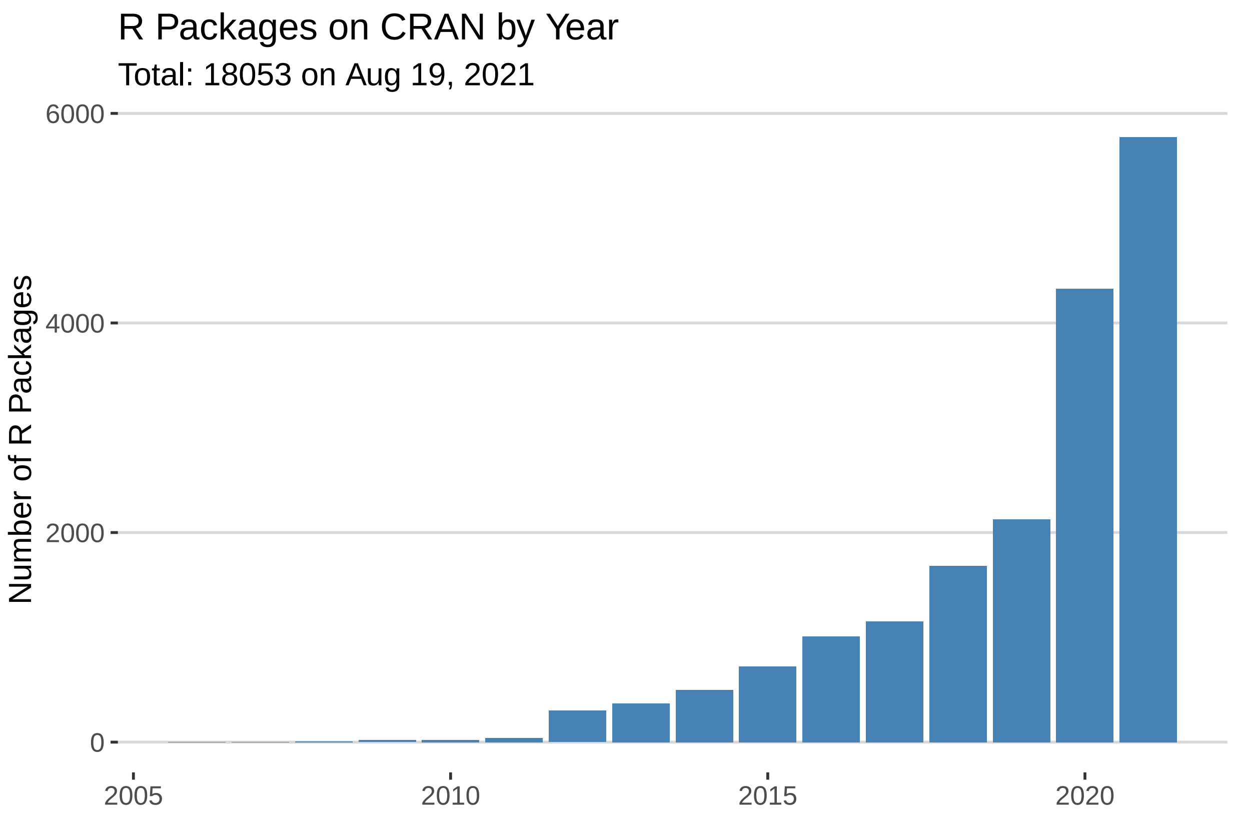
```
install.packages('ggplot2')
```

To load into current R session

```
library(ggplot2)
```

Getting Help

The Universe of R is vast and expanding rapidly as new packages get added to CRAN. The acceleration is evident from the chart below. As a result, it is humanly impossible to learn and remember how to use each func-



tion.

To get help, you could

- Search for answers
  - Access R Help by typing "?" before a function (e.g., ?mean). See Examples
  - Keyword search on Stack Overflow, R Bloggers, or R Mailing List: r-help@r-project.org
  - But, by far the best is Google: (a) copy-paste error messages from R, or (b) keyword search for function followed by R
- Ask for Help
  - Class Discussions Board, Stack Overflow, Github
  - But, be sure to share a reproducible example

When asking for help, clearly state the problem and most importantly, provide a reproducible example of the problem. This may require using a toy dataset (e.g., iris, or mtcars) to illustrate the problem. The `reprex` library offers a handy function that generates code and result in the clipboard to make it easy to paste into Discussions or Stack Overflow. To illustrate, running the function below will generate an output that is ready for pasting into a forum.

```
library(reprex)
reprex(mean(c(1,2,NA,4)),advertise = F)
```

Here is the paste-ready result.

```
mean(c(1, 2, NA, 4))
#> [1] NA
```