Introduction to R

Dr. Noushin Ghaffari

This course will provide an introduction to R. We use the Jupyter Notebook to run the commands. We will also demonstrate command execution using Ada. Parts of this course are based on Software Carpentry "Programming with R" and "R for Reproducible Scientific Analysis" lessons.

What is R?

R is an open source programming language and software environment for statistical computing and graphics that is supported by the R Foundation for Statistical Computing. It supports multiple platforms and can be easily extended.

The “Comprehensive R Archive Network” (CRAN) is a collection of sites which carry identical material, consisting of the R distribution(s), the contributed extensions, documentation for R, and binaries.

The CRAN master site at WU (Wirtschaftsuniversität Wien) in Austria can be found at the URL

```
https://CRAN.R-project.org/
```

and is mirrored daily to many sites around the world. See https://CRAN.R-project.org/mirrors.html (https://CRAN.R-project.org/mirrors.html) for a complete list of mirrors. Please use the CRAN site closest to you to reduce network load [R documetation].

Basic Usage of R

Using R as a Calculator

R can be used as a simple calculator.

```
In [ ]: 100*4.5
```

One should note the order of operations, which affects the results. From highest to lowest precedence:

• Parentheses: (,) • Exponents: ^ or ** • Divide: / • Multiply: * • Add: + • Subtract: -

```
In [ ]: 3*2+10.6
```

```
In [ ]: 3*(2+10.6)
```

```
In [ ]: 3*(2+10.6)/2
In [ ]: 3*2+10.6/2
```

Mathematical Operations in R

R offers many mathematical functions, mainly as part of the “base” package. Here is how you can list all the functions:

```
In [ ]: ?Math
```

```
In [ ]: pi
```

```
In [ ]: sin(90*(pi/180)) #converting radian to degree by pi/180
```

```
In [ ]: log10(10)
```

```
In [ ]: log2(10)
```

```
In [ ]: exp(2.6)
```

Programming with R

Every programming language provides variables, data structure and set of commands. The purpose of these elements is to enable users to write commands that are understandable by the computer and will generate reproducible results. Let's look at R data types:

• character: "R", "test" • numeric: 1, 10, 11.1 • integer: 2L (the L tells R to store this as an integer) • logical: TRUE, FALSE • complex: 1+4i (complex numbers with real and imaginary parts)

```
In [ ]: #Assignment to a variable
        x <- 1:5
```

```
In [ ]: (x)
```

```
In [ ]: a=1
        print(a)
```

```
In [ ]: b=T
        (b)
```

```
In [ ]: c="This is a test"
        print(c)
```

Useful built-in functions to examine features of vectors and other objects, for example

• typeof() - what is the object’s data type (low-level)? • length() - how long is it? What about two dimensional objects?

```
In [ ]: y <- 5:15
        (y)
        typeof(y)
```

In [ ]:
```
length(y)
```

Most important R data structures

- vector (atomic or list)
  - vector is a collection of elements that are most commonly of mode character, logical, integer or numeric
  - Lists can encompass any mixture of data types vs atomic vector that hold only one data type
- matrix
  - In R matrices are an extension of the numeric or character vectors. They are not a separate type of object but simply an atomic vector with dimensions; the number of rows and columns.
- data frame
  - The de facto data structure for most tabular data and what we use for statistics. A data frame is a special type of list where every element of the list has same length

In [ ]:
```
#making a vector
v<- vector("character", length=5)
typeof(v)
```

In [ ]:
```
v<-c("a","b","c","d","e")
```

In [ ]:
```
print(v)
```

In [ ]:
```
#easy addition to the list
v <- c(v,"f","g")
print(v)
```

In [ ]:
```
#Matrix in R
m = matrix(data = 1:10, nrow = 2, ncol = 5)
print(m)
```

In [ ]:
```
nrow(m)
m[1,3] <- 4
(m)
```

In [ ]:
```
ncol(m)
```

In [ ]:
```
m <- cbind(m,33:34)
m
```

In [ ]:
```
#Data frames
df <- data.frame(id = letters[1:10], x = 1:10, y = 11:20)
df
```

In [ ]:
```
names(df)
```

In [ ]:
```
str(df)
```

Working with Data

In [ ]:
```
download.file(url="https://raw.githubusercontent.com/swcarpentry/files/master/inf
lammation-01.csv",destfile="inflammation-01.csv",method="curl")
```

In [ ]:
```
input<-read.csv("inflammation-01.csv",header=F)
```

In [ ]:
```
head(input)
```

In [ ]:
```
getwd()
```

In [ ]:
```
setwd("path_that_you_like")
```

In [ ]:
```
dim(input)
```

In [ ]:
```
input[16,40]
```

In [ ]:
```
input[4,37]
```

In [ ]:
```
input[1:6,3]
```

In [ ]:
```
colnames(input)
```

In [ ]:
```
colnames(input)<- c(paste("C",1:40,sep=''))
```

In [ ]:
```
colnames(input)
```

In [ ]:
```
min(input[,1])
```

In [ ]:
```
max(input[,10])
```

In [ ]:
```
#useful command
str(input)
```

In [ ]:
```
summary(input)
```

In [ ]:
```
avg_day_inflammation <- apply(input, 2, mean)
```

In [ ]:
```
as.data.frame(input$C1)
```

In [ ]:
```
avg_day_inflammation
```

In [ ]:
```
plot(avg_day_inflammation)
```

In [ ]:
```
max_day_inflammation <- apply(input, 2, max)
```

In [ ]:
```
plot(max_day_inflammation)
```

Functions in R

If there are more than one data set to be analyzed and the operations will be repeated, one can write a function so that we can repeat several operations with a single command.

Calculating BMI by writing a function

In [ ]:
```
BMI_Calculator_English_System <- function(weight, height)
{
    BMI <- (weight/ height/ height) *703
    return(BMI)
}
```

In [ ]:
```
BMI_Calculator_English_System(145,60)
```

Loops in R

Loops are useful commands for iterations. In R those can be use to operate items of vectors, matrices and data frames. Lets create a function that uses a “for” loop:

In [ ]:
```
math_ops <- function(a, b)
{
c <- matrix(nrow=nrow(a),ncol=3,data=NA)
colnames(c) <- c("sum","multiply","divid")

for (i in 1:nrow(a))
{
c[i,1] <- a[i] + b[i]
c[i,2] <- a[i] * b[i]
c[i,3] <- a[i] / b[i]
}
    print(c)
}
```

In [ ]:
```
a <- matrix(nrow=4,ncol=1,data=c(1,10,100,50))
```

In [ ]:
```
b <- matrix(nrow=4,ncol=1,c(4,8,12,16))
```

In [ ]:
```
math_ops(a,b)
```

Conditional Statements in R

We use a conditional statement to make a choice based on values of a variable. R syntax for conditional statements is:

if (condition) to start a conditional statement else if (condition) to provide additional tests else to provide a default

The bodies of conditional statements must be surrounded by curly braces {}.

• Use == to test for equality. • X & Y is only true if both X and Y are true. • X | Y is true if either X or Y, or both, are true.

In [ ]:
```
x <- max(input[1,])
```

In [ ]:
```
y <- max(input[15,])
```

In [ ]:
```
if (x > y)
{
  print("x is greater")
} else if (y > x)
{
  print("y is greater")
} else
{
    print("x and y are equal")
}

print(c("x equals to: ",x))
print(c("y equals to: ",y))
```

Let’s write a code that calculates the average population for all the countries in a dataset called "gapminder". We will use a loop and conditional statement to achieve this goal. Let’s download and explore at the first few rows of the data.

In [ ]:
```
download.file("https://raw.githubusercontent.com/swcarpentry/r-novice-gapminder/g
h-pages/_episodes_rmd/data/gapminder-FiveYearData.csv", destfile = "gapminder-Fiv
eYearData.csv",method="curl")
```

In [ ]:
```
gapminder <- read.csv("gapminder-FiveYearData.csv")
```

In [ ]:
```
head(gapminder,25)
```

Calculating the average population for each country and saving the results:

In [ ]:
```
#Making template data structures for output data
gapminder_country_pop_average <- matrix(nrow=1,data=c("country","average pop"))
country_mean <- matrix(nrow=1,ncol=2)

#Initiating the start variable
start <- 1

#For loop through the countries
for (c in 1:(nrow(gapminder)-1))
{
    if(gapminder[c,1]!=gapminder[c+1,1])
    {
        end <- c
        mean <- mean(gapminder[start:end,3])
        country_mean[1,1] <- as.character(gapminder[c,1])
        country_mean[1,2] <- as.numeric(mean)
        gapminder_country_pop_average <- rbind(gapminder_country_pop_average,coun
try_mean)
        start <- c+1
    }

}

#Getting the mean for last country
mean <- mean(gapminder[start:nrow(gapminder),3])
country_mean[1,1] <- as.character(gapminder[c,1])
country_mean[1,2] <- as.numeric(mean)
gapminder_country_pop_average <- rbind(gapminder_country_pop_average,country_mean
)

print(gapminder_country_pop_average)
write.csv(gapminder_country_pop_average,"gapminder_country_pop_average.csv")
```

More sophisticated plots

R can generate complicated plots with high quality. User can customize R functions and take advantage of R packages to generate publication ready plots. Recall the gapmider data that we used in above examples. We will use that dataset to generate informative and sophisticated plots.

In [ ]:
```
str(gapminder)
```

The str function shows that gapminder data includes country: factor with 142 levels continent: factor with 5 levels year: ranges from 1952 to 2007 in increments of 5 years lifeExp: life expectancy at birth, in years pop: population gdpPercap: GDP per capita, where GDP is gross domestic product.

In this example, we load a package called "ggplot2" to create our plots.

In [ ]:
```
library("ggplot2")
```

In [ ]:
```
ggplot(data = gapminder, aes(x = gdpPercap, y = lifeExp)) +   geom_point()
```

In [ ]:
```
ggplot(data = gapminder, aes(x=year, y=lifeExp, by=country, color=continent)) +
geom_line()
```

Useful Commands

match() sessionInfo() ls() list.files()

In [ ]:
```
#useful function especially to be added at the end of your functions to save the exact date and time
#of running the function
date()
```

In [ ]:
```
(a <- 1:10)
(b <- 5:14)
print(match(a,b))
```

In [ ]:
```
sessionInfo()
```

In [ ]:
```
ls()
```

In [ ]:
```
list.files()
```

R Packages

It is possible to add functions to R by writing a package, or by obtaining a package written by someone else. As of this writing, there are over 10,000 packages available on CRAN (the comprehensive R archive network). R and RStudio have functionality for managing packages:

- You can see what packages are installed by typing installed.packages()
- You can install packages by typing install.packages("packagename"), where packagename is the package name, in quotes.
- You can update installed packages by typing update.packages()
- You can remove a package with remove.packages("packagename")
- You can make a package available for use with library(packagename)