Essential R

R. Nazim Khan

September 9, 2022

1 Introduction

This is a basic introdcution to R a statistical environment. More information can be found in a many references. One particular complete reference is [1].

2 Data Basics

In this section we discuss basic data manipulation. We also discuss how to read data into R and write them out to files.

2.1 Data manipulation

The basic data types in R are numerical, characters, strings and nominal/ordinal. Each data type can be stored as constants, vectors, matrices or dataframes. There are several ways to read data into R. Data can be directly entered into R by defining variables.

```
a<-1.2
b<-pi
x<-c(1:5)
y<-c(6:10)
x

## [1] 1 2 3 4 5

y

## [1]  6  7  8  9 10

z<-c(1.1,2.1,3.2,4.5,2.4,3.8,4.3)
z

## [1] 1.1 2.1 3.2 4.5 2.4 3.8 4.3

z[2]
```

```
## [1] 2.1

z[1:2]

## [1] 1.1 2.1

z[c(2,4)]

## [1] 2.1 4.5

which(z>4)

## [1] 4 7

z[z<4]

## [1] 1.1 2.1 3.2 2.4 3.8

z[-2]

## [1] 1.1 3.2 4.5 2.4 3.8 4.3

#This lists z without its second element.
```

First the constants $a = 1.2$ and $b = \pi$ have been defined. The vector `x` contains the numbers 1 to 5. Note that the syntax `1:5` indicates integers between 1 to 5 inclusive. The vector z contains real numbers.

```
x*y

## [1]  6 14 24 36 50

x+y

## [1]  7  9 11 13 15

x*z

## Warning in x * z:  longer object length is not a multiple of shorter object
length

## [1]  1.1  4.2  9.6 18.0 12.0  3.8  8.6

y%%2

## [1] 0 1 0 1 0

3%/%2

## [1] 1
```

The symbol + indicated addition, $*$ indicates multiplication while / indicated real division, and these are performed term by term. If the objects are not of the same length then the shorter one is re-cycled to match the lengths. The symbol %% represents modulo division, while %/% represents integer division.

More complicated computations can be performed:

```
(x+y)^2
```

```
## [1]  49  81 121 169 225
```

```
log(x)
```

```
## [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
```

```
log10(x)
```

```
## [1] 0.0000000 0.3010300 0.4771213 0.6020600 0.6989700
```

```
log2(x)
```

```
## [1] 0.000000 1.000000 1.584963 2.000000 2.321928
```

```
logb(x,3)
```

```
## [1] 0.0000000 0.6309298 1.0000000 1.2618595 1.4649735
```

```
exp(x)
```

```
## [1]   2.718282   7.389056  20.085537  54.598150 148.413159
```

```
10^x
```

```
## [1] 1e+01 1e+02 1e+03 1e+04 1e+05
```

Note that log represents logarithm to the base e.

A large range of mathematical functions is available.

```
sin(pi)
```

```
## [1] 1.224606e-16
```

```
exp(-1)
```

```
## [1] 0.3678794
```

```
log(exp(1))
```

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

Note that R can also handle complex numbers.

```
x<-(1+2i)
y<-(3+4i)
x+y

## [1] 4+6i

x^2

## [1] -3+4i

x^y

## [1] 0.1290096+0.0339241i

sqrt(x)

## [1] 1.27202+0.786151i

sqrt((-1+0i))

## [1] 0+1i
```

Character types can also be easily handled by R.

```
x<-c("Hummer","Dodge","Bentley")
y<-c("Toyota","Honda","Mitsubishi")
str(x)

##  chr [1:3] "Hummer" "Dodge" "Bentley"
```

Exercises

1. Sales ($millions) for a chain of stores for last year and this year for the month of January, in store correspondence are:

   last years: 1.5, 1.7,2.1,3.4,1.3,2.4,4.5,0.9
   this years: 1.6, 1.8,1.9,3.5,1.1,2.2,4.5, 1.2

   Is there a difference in the mean sales between January last year and this year? Write the R code to compute the appropriate test statistic to test the appropriate hypotheses.

2. Write the R code to compute the variance of the combined sales figures in the previous exercise.

3. Suppose the sales were for a random sample of stores from a franchise. Now compute the appropriate test statistic to determine if the sales have increased from last year.

Matrices are read as follows.

```
x<-matrix(nrow=3,c(1,2,3,4,5,6,7,8,9),byrow=F)
x

##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9

class(x)

## [1] "matrix" "array"

attributes(x)

## $dim
## [1] 3 3

x[1,]

## [1] 1 4 7

x[,1]

## [1] 1 2 3

summary(x)

##        V1            V2            V3
##  Min.   :1.0   Min.   :4.0   Min.   :7.0
##  1st Qu.:1.5   1st Qu.:4.5   1st Qu.:7.5
##  Median :2.0   Median :5.0   Median :8.0
##  Mean   :2.0   Mean   :5.0   Mean   :8.0
##  3rd Qu.:2.5   3rd Qu.:5.5   3rd Qu.:8.5
##  Max.   :3.0   Max.   :6.0   Max.   :9.0
```

Another very useful function is `gl()`, which creates a factor with the specified number of levels. The basic syntax is

```
Nitrogen<-gl(n= 3, k=10, length = 30, labels = c("Low","Med","High"), ordered = T)
class(Nitrogen)

## [1] "ordered" "factor"
```

```
is.factor(Nitrogen)

## [1] TRUE

is.ordered(Nitrogen)

## [1] TRUE
```

where `n` give the number of levels, `k` gives the number of times each is repeated, `length` gives the option of repeating this set, `labels` provides a set of optional levels, and `ordered` specifies whether the factor is ordered.

2.2 Logical comparisons

`R` understands logical comparisons $<, >, <=, >=$, which are applied elementwise. Note that logical equality is $==$ and inequality is $!=$, while & is 'logical and', | is 'logical or'.

```
(1:5) == (5:1)

## [1] FALSE FALSE  TRUE FALSE FALSE

(1:5)>(5:1)

## [1] FALSE FALSE FALSE  TRUE  TRUE

((1:5)==(5:1))|((1:5)>(5:1))

## [1] FALSE FALSE  TRUE  TRUE  TRUE

((1:5)==(5:1))& ((1:5)<(5:1))

## [1] FALSE FALSE FALSE FALSE FALSE
```

Use `help("!")` to obtain information regarding logical operators. The following functions are also useful.

`any()` Returns `TRUE` if any of the argument satisfies the criteria.

`all()` Returns `TRUE` if all of the argument satisfies the criteria.

`identical()` Returns `TRUE` if the two objects are exactly the same.

`all.equal()` Returns `TRUE` the two objects are (near) equal.

```
identical(sin(pi),0)

## [1] FALSE
```

```r
all.equal(sin(pi),0)

## [1] TRUE

x<-c(1:4,-1,-3,0)
if(any(x<0)) cat("Some x are negative")

## Some x are negative
```

2.3 Changing the working directory

The working directory can be changed from `File` ⇒ `Change dir...`. This then gives a dialogue box that allows one to browse for the required directory.

2.4 Reading data into `R`

There are various ways to read data into `R` depending on the data format.

1. For files ending in `.R` or `.r` use `source()`.
2. For files ending in `.Rdata` or `.rda` use `load()`.
3. For files ending in `.tab`, `.txt` or `.TXT` use `read.table()`, which produces a dataframe.
4. For files ending in .csv or `.CSV` use `read.table(...,header=TRUE, sep=";")`, which produces a dataframe.

2.4.1 The function `scan`

The function `scan` can be used to read data into `R` from a file. The basic syntax is

```
scan(file = "filename", sep = "",
     skip = 0, nlines = 0, na.strings = "NA")
```

If the filename is "" then the input is from the keyboard (or the `stdin`), and is terminated by a blank line. Only one column of data can be entered this way.

2.4.2 The function `read.table`

This function is used to read a dataframe from a file. The basic syntax is

```
read.table(file, header = FALSE, sep = "", quote = "\",...)
```

Typically the file is a `.txt` or `.csv`. The separator is usually a space `sep=" "` in which case it can be omitted, a tab `sep="\t"`, a comma `sep=","` or a semicolon `sep=";"`. If `header=T` then column names will be read in from the first line of the file. Otherwise column names may be specified by a vector. Similarly, row names may also be specified by a vector.

There are related functions `read.csv()`, `read.delim()` and a few variants. Use the `help` function for more details. Excel files can also be read in—see the library `readxl` for details.

2.4.3 Accessing built in data sets

There are several data sets supplied with R, and several that come with different packages. All datasets supplied with R are directly available by name.

2.5 Dataframes

Several columns together form a data set, with each row containing a **record**, that is, observations on a single **experimental unit**. In R such an object is called a **dataframe**. The matrix class and the dataframe class are very similar in R except that some operations can be performed on matrices but not on dataframes. The dataframe will have column names that are used to refer to the variables. In R the dataframe may also have row names defined. Row names may be used to refer to each record, and this is meaningful in many contexts. For example, the observations may be demographic information on cities, and the records may be identified by the city names which are stored as row names.

A dataframe is defined by the functions `data.frame()` or `as.data.frame()`. Below we create a simple data frame.

```
Make<-c("Honda","Chevrolet","Ford","Eagle","Volkswagen","Buick","Mitsbusihi","Dodge","Chrys
Model<-c("Civic","Beretta","Escort","Summit","Jetta","Le Sabre","Galant","Grand Caravan","N
Cylinder<-c(rep("V4",5),"V6","V4",rep("V6",3))
Weight<-c(2170,2655,2345,2560,2330,3325,2745,3735,3450,3265)
Mileage<-c(33,26,33,33,26,23,25,18,22,20)
Type<-c("Sporty","Compact",rep("Small",3),"Large","Compact","Van",rep("Medium",2))
Car<-data.frame(Make,Model,Cylinder,Weight,Mileage,Type)
Car
```

```
##          Make         Model Cylinder Weight Mileage    Type
## 1       Honda         Civic       V4   2170      33  Sporty
## 2   Chevrolet       Beretta       V4   2655      26 Compact
## 3        Ford        Escort       V4   2345      33   Small
## 4       Eagle        Summit       V4   2560      33   Small
## 5  Volkswagen         Jetta       V4   2330      26   Small
## 6       Buick      Le Sabre       V6   3325      23   Large
## 7  Mitsbusihi        Galant       V4   2745      25 Compact
## 8       Dodge Grand Caravan       V6   3735      18     Van
## 9    Chrysler    New Yorker       V6   3450      22  Medium
## 10      Acura        Legend       V6   3265      20  Medium
```

```
Car[1,]
```

```
##    Make Model Cylinder Weight Mileage   Type
## 1 Honda Civic       V4   2170      33 Sporty
```

```
Car[,1]
```

```
##  [1] "Honda"      "Chevrolet"  "Ford"       "Eagle"      "Volkswagen" "Buick"      "Mits
```

```
Car$Model
```

```
##  [1] "Civic"         "Beretta"       "Escort"        "Summit"        "Jetta"          "Le
##  [9] "New Yorker"    "Legend"
```

```
table(Car$Type)
```

```
##
## Compact   Large  Medium   Small  Sporty     Van
##       2       1       2       3       1       1
```

The proportion of cars of each type can be produced by:

```
table(Car$Type)/length(Car$Type)
```

```
##
## Compact   Large  Medium   Small  Sporty     Van
##     0.2     0.1     0.2     0.3     0.1     0.1
```

Cross tables can also be produced easily.

```
table(Car$Make, Car$Type)
```

```
##
##              Compact Large Medium Small Sporty Van
##   Acura            0     0      1     0      0   0
##   Buick            0     1      0     0      0   0
##   Chevrolet        1     0      0     0      0   0
##   Chrysler         0     0      1     0      0   0
##   Dodge            0     0      0     0      0   1
##   Eagle            0     0      0     1      0   0
##   Ford             0     0      0     1      0   0
##   Honda            0     0      0     0      1   0
##   Mitsbusihi       1     0      0     0      0   0
##   Volkswagen       0     0      0     1      0   0
```

The dataframe can also be sorted by any variable. For example, below the dataframe is sorted by weight.

```
i<-order(Car$Weight);i
```

```
##  [1]  1  5  3  4  2  7 10  6  9  8
```

```
Car[i,]
```

```
##          Make         Model Cylinder Weight Mileage    Type
## 1       Honda         Civic       V4   2170      33  Sporty
## 5  Volkswagen         Jetta       V4   2330      26   Small
## 3        Ford        Escort       V4   2345      33   Small
## 4       Eagle        Summit       V4   2560      33   Small
## 2   Chevrolet       Beretta       V4   2655      26 Compact
## 7  Mitsbusihi        Galant       V4   2745      25 Compact
## 10      Acura        Legend       V6   3265      20  Medium
## 6       Buick      Le Sabre       V6   3325      23   Large
## 9    Chrysler    New Yorker       V6   3450      22  Medium
## 8       Dodge Grand Caravan       V6   3735      18     Van
```

Note that in the Windows platform, dataframes can be accessed directly through `Edit`→Data editor *ldots*, or the command `data1<-edit(data.frame())`. A spreadsheet is produced in which data can be entered directly. However, this requires care to ensure that the data types are defined correctly. This is especially critical for factors.

2.6 Writing data to files

The function `write(x, file = "data",ncolumns = if(is.character(x)) 1 else 5, append = FALSE, sep = " ")` writes `x`, usually a matrix, to a file. If `append=TRUE)` then the data is appended at the end of the file. A related function is

`write.table(x, file = "", append = FALSE, sep = " ",dec = ".", row.names = TRUE, col.names = TRUE)`

used to write dataframes to file. If the object to be written is not a data frame then it is coerced to be one.

2.7 `R` script files

When writing a long piece of code it is easy to make mistakes, and the whole code needs to be typed in again. In addition, some code may be used again. One should *always* write the code in a file and then run it in `R`. Simply access the `R Editor` from `File` ⇒ `New File` ⇒ `R Script`and saving it with a `.R` extension by default. Such a file can then be run in `R` , and also used as a template to write new code.

When building a long piece of code, one should enter the code a few lines at a time and test it before continuing.

Exercises

1. The formula for computing the interest paid on a loan of \$1,000 compounded annually if the nominal annual rate is 7.5% is

$$\text{Interest} = 1000\left((1+0.075)^5 - 1\right)$$

   (a) Write the `R` code for the above expression.

   (b) What is the result of your computation above?

(c) Modify the expression to determine the amount of interest paid if the nominal annual rate is 3.5%.

(d) What happens if the exponent 5 is replaced by `(1:10)`?

2. Write `R` code that prints out the perfect squares up to and including 100.

References

[1] Michael J. Crawley, *The R Book*, John Wiley & Sons, Ltd, West Sussex, UK, 2nd edition, 2012.