**A Quick Introduction to R**

Download R from a CRAN Mirror: Current Version is 4.02.2 for windows

Also, available for Mac and Linux.

Download RStudio from <https://rstudio.com/products/rstudio/download/#download>

This is the free version

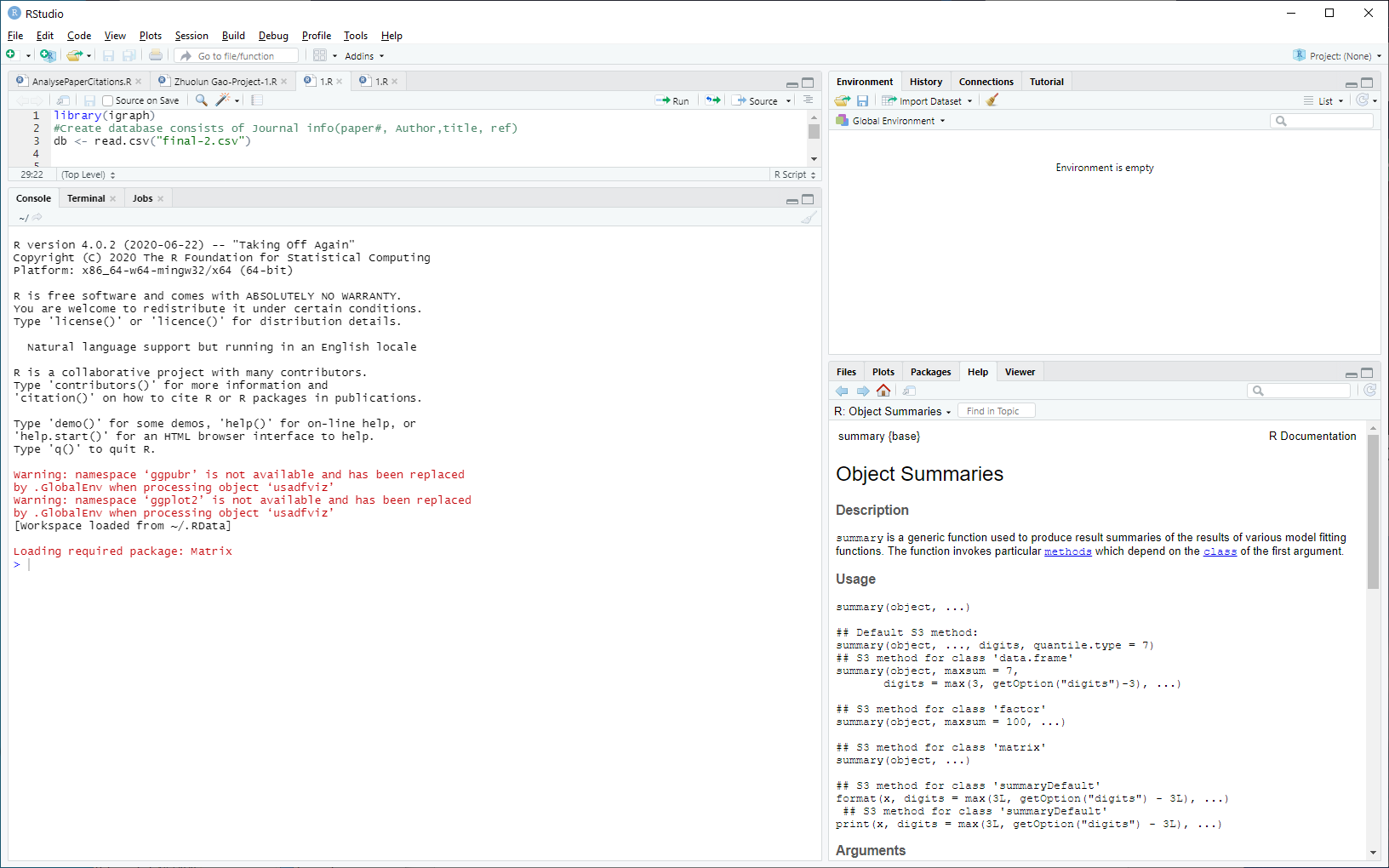
Current version is RStudio Desktop 1.3.1073

See also <https://docs.rstudio.com/>.

Run the executables for both.

Start RStudio.

You will see a four-pane screen:



You enter commands into the lower left pane – the Console.

The upper right pane is the current contents of the workspace.

The lower right pan has multiple functions which we will see.

Click on the Edit Menu and select “Clear Console”

Click on the Session Menu and select “Clear Workspace”

**R** is a command line oriented program.

You will make a lot of typing mistakes.

Thus, use RStudio which provides a text editor

Use the First() function to configure your environment upon startup.

Use the initialization files:

Put in .Rprofile for local configuration or Rprofile for global configuration.

For these methods, read up on them in one of the introductory R books.

Setting/Getting working directory:

setwd(“g:\\SHKDir”);

getwd();



Changing to my working directory for this intro



During an R session, all objects are stored in a temporary, working memory – the *workspace*.

List objects



Remove objects

rm()

Objects that you want to access later must be saved in the workspace.

from the menu bar: File->save workspace

from the command line: save(x,file=“MyData.Rdata”)

Variable Naming conventions:

must start with a letter (A-Z or a-z)

can contain letters, digits (0-9), and/or periods “.”

case-sensitive

mydata different from MyData

do not use underscore “\_”

Command line history

Can be saved and loaded

savehistory(file=“MyData.Rhistory”)

loadhistory(file=“MyData.Rhistory”)

Display history

history(max.show=Inf)

During a session you can use the arrow keys to review the command history; these navigate up and down within the console.

When you clear the console, the current history is deleted.

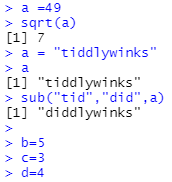
Pick one of the R books: Read the section on variables, constants, expressions, etc. These syntactic and semantic structures are very similar to Java.

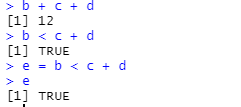
Assign a value to a variable – the variable takes on the type of the value.

First, a is an integer, then a is a string.

The function sub performs a substitution and returns the new value.

But, the value of a is still “tiddlywinks”





TRUE and FALSE are the Boolean values.

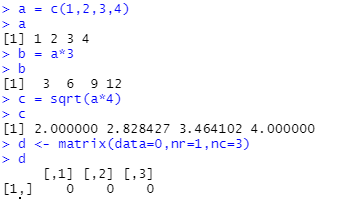
R supports vectors, matrices, and arrays.

a is a vector or a 1x4 matrix.

You can multiply the entire contents of a vector, matrix, or array by a constant

Or you can apply a function too every element of one of these.

And, you can create 2-D matrices and initialize the contents.



In R, a single number is the special case of a vector with 1 element.

Other vector types: character strings, logical values

A vector: x=c(100,101,102,103) Notation: [ ] is used to access elements in x

A vector of logical values:

> c(TRUE, FALSE, TRUE, FALSE, FALSE)   
[1]  TRUE FALSE  TRUE FALSE FALSE

Extract 2nd element in x  
> x[2]  
[1] 101

Finding the length of a vector:

> length(c("aa", "bb", "cc", "dd", "ee"))   
[1] 5

Extract 3rd and 4th elements in x   
> x[3:4] # or x[c(3,4)]  
[1] 102 103

> x   
[1] 100 101 102 103

Arithmetic Operations:

> a = c(1, 3, 5, 7)

> b = c(1, 2, 4, 8)

> 5 \* a   
[1]  5 15 25 35

> a + b   
[1]  2  5  9 15

> a / b   
[1] 1.000 1.500 1.250 0.875

> u = c(10, 20, 30)   
> v = c(1, 2, 3, 4, 5, 6, 7, 8, 9)   
> u + v   
[1] 11 22 33 14 25 36 17 28 39

Note: Shorter vector is recycled to match longer vector!

A new vector can be sliced from an existing vector:

> s = c("aa", "bb", "cc", "dd", "ee")

Define logical vector L

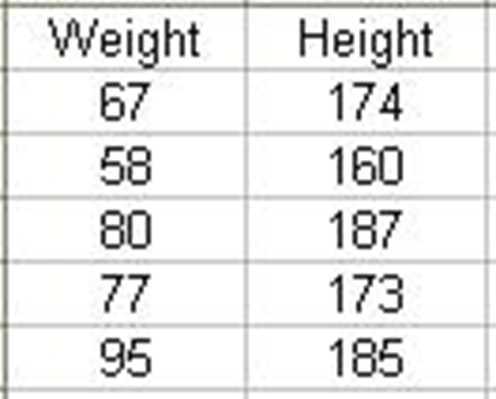
> L = c(FALSE, TRUE, FALSE, TRUE, FALSE)   
> s[L]   
[1] "bb" "dd"

> s[c(FALSE, TRUE, FALSE, TRUE, FALSE)]   
[1] "bb" "dd"

A matrix is a table of different classes

Each **column** must be of the **same** class (e.g. numeric, character, etc.)

The number of elements in each row must be identical.



Accessing elements in matrices:

**x[row,column]**

The ‘Height’ column:  
**> x[,”Height”] # or:   
> x[,2]**

Note: you **cannot** use “**$**”  
**> x$Weight**

> B = matrix(  c(2, 4, 3, 1, 5, 7),  nrow=3, ncol=2)   
   
> B             # B has 3 rows and 2 columns   
      [,1] [,2]   
[1,]    2    1   
[2,]    4    5   
[3,]    3    7

> t(B)          # transpose of B   
      [,1] [,2] [,3]   
[1,]    2    4    3   
[2,] 1 5 7

> C = matrix( c(7, 4, 2), nrow=3, ncol=1)

Combine B and C

> cbind(B, C)   
     [,1] [,2] [,3]   
[1,]    2    1    7   
[2,]    4    5    4   
[3,]    3    7    2

Combine the rows of two matrices if they have the same number of columns with the rbind function.

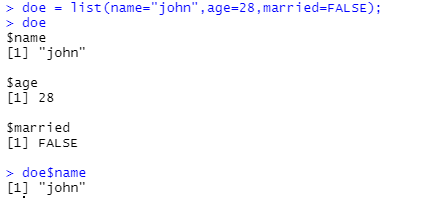
> D = matrix(c(6, 2), nrow=1, ncol=2)   
> D             # D has 2 columns   
     [,1] [,2]   
[1,]    6    2   
   
> rbind(B, D)   
     [,1] [,2]   
[1,]    2    1   
[2,]    4    5   
[3,]    3    7   
[4,]    6

A list is an ordered collection of data of arbitrary types.

Create the list doe.

When printing doe, it prints the name of each field and its value.

You can access fields by name.



Typically, vector elements are accessed by their index (an integer), list elements by their name (a character string).

But both types support both access methods.

Another type of structure is a *data frame*: like a table with row and column labels.

Different columns may have different data types.

> a

localisation tumorsize progress

XX348 proximal 6.3 FALSE

XX234 distal 8.0 TRUE

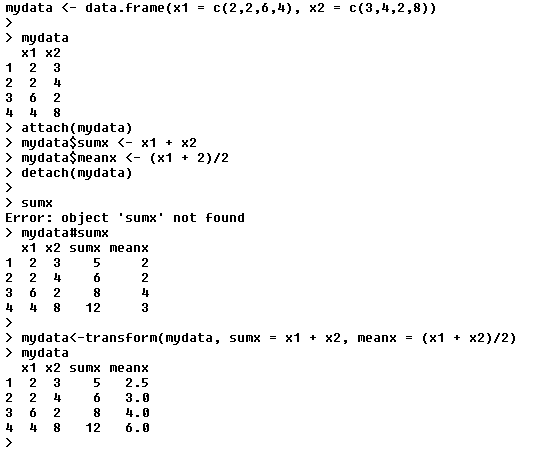
XX987 proximal 10.0 FALSE

With this code, you obtain the original data frame and

two vectors: sumx and meanx – added into the data frame

BUT: note the rounding in meanx!

So, use this form to force to Floating point



A character string can contain arbitrary text.

* Sometimes it is useful to use a limited vocabulary, with a small number of allowed words.
* A factor is a variable that can only take such a limited number of values, which are called levels.

> a

[1] Kolon(Rektum) Magen Magen

[4] Magen Magen Retroperitoneal

[7] Magen Magen(retrogastral) Magen

Levels: Kolon(Rektum) Magen Magen(retrogastral) Retroperitoneal

> class(a)

[1] "factor"

> as.character(a)

[1] "Kolon(Rektum)" "Magen" "Magen"

[4] "Magen" "Magen" "Retroperitoneal"

[7] "Magen" "Magen(retrogastral)" "Magen"

> as.integer(a)

[1] 1 2 2 2 2 4 2 3 2

> as.integer(as.character(a))

[1] NA NA NA NA NA NA NA NA NA NA NA NA

Warning message: NAs introduced by coercion, means “not applicable“

Note: ‘ as‘ allows you to display in different formats

You may extract subsets of vectors, matrices, and arrays, and data frames:

Individual elements are accessed with “[ ]” by specifying their index, or their name

> a[3, 2]

[1] 10

> a["XX987", "tumorsize"]

[1] 10

> a["XX987",]

localisation tumorsize progress

XX987 proximal 10 0

> a[c(1,3),] ; subset rows by a vector of indices

localisation tumorsize progress

XX348 proximal 6.3 0

XX987 proximal 10.0 0

Note: The “,’ with no index means “display all coumns”.

> a[c(T,F,T),] ; subset rows by a logical vector

localisation tumorsize progress

XX348 proximal 6.3 0

XX987 proximal 10.0 0

> a$localisation ; subset a column

[1] "proximal" "distal" "proximal„

> a$localisation=="proximal“

> ; comparison resulting in logical vector - “;’ indicates thereafter is a comment

[1] TRUE FALSE TRUE

> a$localisation=="proximal", ]

> ; subset the selected rows

localisation tumorsize progress

XX348 proximal 6.3 0

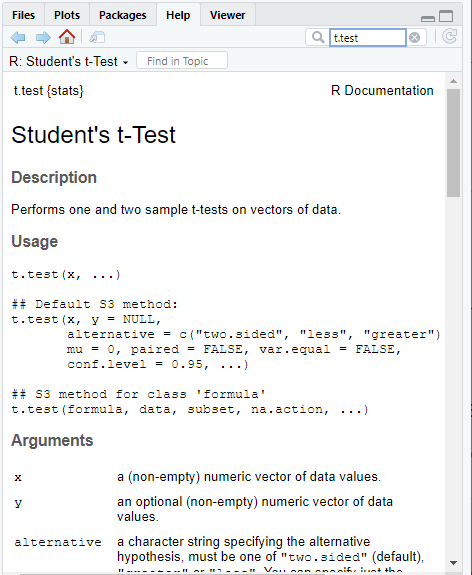
XX987 proximal 10.0 0

Getting help for a function name

>help(t.test)

starting httpd help server ... Done

In a separate browser window:



Grouped expressions

R is an expression language in the sense that its only command type is a function or expression which returns a result.

Commands may be grouped together in braces, {expr 1, . . ., expr m}, in which case the value of the group is the result of the last expression in the group evaluated.

Control statements

if statements

The language has available a conditional construction of the form

if (expr 1) expr 2 else expr

where expr 1 must evaluate to a logical value and the result of the entire expression is then evident.

a vectorized version of the if/else construct, the ifelse function.

ifelse(condition, a, b)

Repetitive Execution:

for loops, repeat and while

*for (name in expr 1) expr 2*

where name is the loop variable.

expr 1 is a vector expression, (often a sequence like 1:20), and

expr 2 is often a grouped expression with its sub-expressions written in terms of the dummy name.

expr 2 is repeatedly evaluated as name ranges through the values in the vector result of expr 1.

*repeat expr* statement

*while (condition) expr* statement.

The *break* statement can be used to terminate any loop, possibly abnormally.

The only way to terminate repeat loops.

The *next* statement can be used to discontinue one particular cycle and skip to the “next”.

Importing Data:

read.table()

reads in data from an external file

data.entry()

create object first, then enter data

c()

concatenate

scan()

prompted data entry

start editor and save changes

data.entry(x)

start editor, changes not saved

de(x)

start text editor

edit(x)

Defining a function:

Suppose you want to multiply a fractional number by 100, round to one decimal place, and print the number with a % after the rounded number.

addPercent <- function(x)

{

percent <- round(x \* 100, digits = 1)

result <- paste(percent, “%’, sep = “”)

return(result)

}

Note:

-addPercent is the name of the function

-function must always be followed by (), possibly with parameters

-body of function is enclosed in {…}

-return(…) is used to return a result

As noted, R has thousands of packages for almost every conceivable kind of computation.

You can search the list of available packages on a CRAN Mirror.

You must install a package in order to use. This means loading it into you workspace.

install.packages("readr") <- Note double quotes “”

also installing the dependencies ‘Rcpp’, ‘curl’, ‘BH’

trying URL 'http://cran.rstudio.com/bin/windows/contrib/3.1/Rcpp\_0.12.0.zip‘

Content type 'application/zip' length 3131494 bytes (3.0 Mb)

opened URL

downloaded 3.0 Mb

trying URL 'http://cran.rstudio.com/bin/windows/contrib/3.1/curl\_0.9.3.zip‘

Content type 'application/zip' length 4384774 bytes (4.2 Mb)

opened URL

downloaded 4.2 Mb

trying URL 'http://cran.rstudio.com/bin/windows/contrib/3.1/BH\_1.58.0-1.zip‘

Content type 'application/zip' length 13846661 bytes (13.2 Mb)

opened URL

downloaded 13.2 Mb

trying URL 'http://cran.rstudio.com/bin/windows/contrib/3.1/readr\_0.1.1.zip‘

Content type 'application/zip' length 1128326 bytes (1.1 Mb)

opened URL

downloaded 1.1 Mb

package ‘Rcpp’ successfully unpacked and MD5 sums checked

package ‘curl’ successfully unpacked and MD5 sums checked

Then, you must declare it in order for R to know that it can use it.

library(readr) **<- Note no double quotes “”**

Warning message: package ‘readr’ was built under R version 3.1.3

contents <- read\_file("h:/Rdata/test1.txt")

Contents

[1] "Message-ID: <19874053.1075845747143.JavaMail.evans@thyme>\r\nDate: Tue, 13 Jun 2000 00:50:00 -0700 (PDT)\r\nFrom: gregg.penman@enron.com\r\nTo: kay.mann@enron.com\r\nSubject: Re: Revised RM Policy\r\nMime-Version: 1.0\r\nContent-Type: text/plain; charset=us-ascii\r\nContent-Transfer-Encoding: 7bit\r\nX-From: Gregg Penman\r\nX-To: Kay Mann\r\nX-cc: \r\nX-bcc: \r\nX-Folder: \\Kay\_Mann\_June2001\_1\\Notes Folders\\Chicago\r\nX-Origin: MANN-K\r\nX-FileName: kmann.nsf\r\n\r\nPer my voice mail, will you please take a quick look at my edits to Vlady's \r\ndraft risk management policy attached below. I don't think any of them are \r\nproblematic, but Vlady has some concerns mainly because the base Enron Corp. \r\nrisk management policy was used as the template and therefore I am \r\neffectively making edits to it.\r\n\r\nI will be leaving the office today around 11:00 so I want to turn this to \r\nPeoples before then if possible. On any other business, I will be \r\nunavailable from 1:30 to probably 3:00 today, but can be reached via pager \r\n800-980-1771 or cell phone 312-925-8888 at any other time.\r\n\r\nAs always - thanks for your help.\r\n\r\nGregg\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n

See: http://blog.rstudio.org/2015/04/09/readr-0-1-0/

This is what the file’s contents look like:

Message-ID: <19874053.1075845747143.JavaMail.evans@thyme>

Date: Tue, 13 Jun 2000 00:50:00 -0700 (PDT)

From: gregg.penman@enron.com

To: kay.mann@enron.com

Subject: Re: Revised RM Policy

Mime-Version: 1.0

Content-Type: text/plain; charset=us-ascii

Content-Transfer-Encoding: 7bit

X-From: Gregg Penman

X-To: Kay Mann

X-cc:

X-bcc:

X-Folder: \Kay\_Mann\_June2001\_1\Notes Folders\Chicago

X-Origin: MANN-K

X-FileName: kmann.nsf

Per my voice mail, will you please take a quick look at my edits to Vlady's

draft risk management policy attached below. I don't think any of them are

problematic, but Vlady has some concerns mainly because the base Enron Corp.

risk management policy was used as the template and therefore I am

effectively making edits to it.

I will be leaving the office today around 11:00 so I want to turn this to

Peoples before then if possible. On any other business, I will be

unavailable from 1:30 to probably 3:00 today, but can be reached via pager

800-980-1771 or cell phone 312-925-8888 at any other time.

As always - thanks for your help.

Gregg

So, you can use read\_lines to get each line to be a row in a data frame:

line1<-read\_lines("h:/Rdata/test1.txt")

Line1

[1] "Message-ID: <19874053.1075845747143.JavaMail.evans@thyme>“

[2] "Date: Tue, 13 Jun 2000 00:50:00 -0700 (PDT)“

[3] "From: [gregg.penman@enron.com](mailto:gregg.penman@enron.com)“

[4] "To: [kay.mann@enron.com](mailto:kay.mann@enron.com)“

[5] "Subject: Re: Revised RM Policy“

[6] "Mime-Version: 1.0“

[7] "Content-Type: text/plain; charset=us-ascii“

[8] "Content-Transfer-Encoding: 7bit“

[9] "X-From: Gregg Penman“

[10] "X-To: Kay Mann“

[11] "X-cc: “

[12] "X-bcc: “

[13] "X-Folder: \\Kay\_Mann\_June2001\_1\\Notes Folders\\Chicago“

[14] "X-Origin: MANN-K“

[15] "X-FileName: kmann.nsf“

[16] "“

[17] "Per my voice mail, will you please take a quick look at my edits to Vlady's “

[18] "draft risk management policy attached below. I don't think any of them are “

[19] "problematic, but Vlady has some concerns mainly because the base Enron Corp. “

[20] "risk management policy was used as the template and therefore I am “

[21] "effectively making edits to it.“

[22] "“

[23] "I will be leaving the office today around 11:00 so I want to turn this to “

[24] "Peoples before then if possible. On any other business, I will be “

[25] "unavailable from 1:30 to probably 3:00 today, but can be reached via pager “

[26] "800-980-1771 or cell phone 312-925-8888 at any other time.“

[27] "“

[28] "As always - thanks for your help.“

[29] "“

[30] "Gregg“

Extracting Strings:

Suppose we need to extract the name from an email address

>email<-[colleen.koenig@enron.com](mailto:colleen.koenig@enron.com)

>parts<-str\_sub(email,0,str\_locate(email,”@”)-1)

>parts

[1] “colleen.koenig” “colleen.koenig”

>length(parts)

[1] 2

>parts[[1]]

[1] “colleen.koenig”

Then extract the parts of the name.

Character Vectors:

Character strings are always double quoted

Vectors made of character strings:

> x=c("I","want","to","go","home")   
> x   
[1] "I" "want" "to" "go" "home"

Using rep():  
> rep("bye",2)   
[1] "bye" "bye"

Notice the difference using paste() (1 element):  
> paste("I","want","to","go","home")  
[1] "I want to go home"

**R: Workspaces**

* During an R session, all objects are stored in a temporary, working memory – the *workspace*.
* List objects
  + ls()
* Remove objects
  + rm()
* Objects that you want to access later must be saved in the workspace.
  + from the menu bar: File->save workspace
  + from the command line: save(x,file=“MyData.Rdata”)
* Variable Naming conventions:
  + must start with a letter (A-Z or a-z)
  + can contain letters, digits (0-9), and/or periods “.”
    - case-sensitive
    - mydata different from MyData
    - do not use underscore “\_”

**R: History**

* Command line history
* Can be saved and loaded
  + savehistory(file=“MyData.Rhistory”)
  + loadhistory(file=“MyData.Rhistory”)
* Display history
  + history(max.show=Inf)
* During a session you can use the arrow keys to review the command history

**R: Find the Factorial of a Number**

# take input from the user

num = as.integer(readline(prompt="Enter a number: "))

factorial = 1

# check is the number is negative, positive or zero

If (num < 0)

{

print("Sorry, factorial does not exist for negative numbers")

}

else if(num == 0)

{

print("The factorial of 0 is 1")

}

else

{

for(i in 1:num)

{

factorial = factorial \* i

}

}

print(paste("The factorial of", num ,"is",factorial))

**R: Find The GCD of Two Numbers**

# Program to find the H.C.F of two input number

#The greatest common divisor (G.C.D) of two numbers is the largest positive integer that #perfectly divides the two given numbers.

# define a function

hcf <- function(x, y)

{

# choose the smaller number

if(x > y)

{

smaller = y

}

else

{

smaller = x

}

for(i in 1:smaller)

{

if((x %% i == 0) && (y %% i == 0))

{

hcf = i

}

}

return(hcf)

}

# take input from the user

num1 = as.integer(readline(prompt = "Enter first number: "))

num2 = as.integer(readline(prompt = "Enter second number: "))

print(paste("The H.C.F. of", num1,"and", num2,"is", hcf(num1, num2)))

**R: Check If a Prime Number**

# A positive integer greater than 1 which has no other factors except 1 and the number itself is

# called a prime number.

# Program to check if the input number is prime or not

# take input from the user

num = as.integer(readline(prompt="Enter a number: "))

flag = 0

# prime numbers are greater than 1

if(num > 1)

{

# check for factors

flag = 1

for(i in 2:(num-1))

{

if ((num %% i) == 0)

{

flag = 0

break

}

}

}

if(num == 2) flag = 1

if(flag == 1)

{

print(paste(num,"is a prime number"))

}

else

{

print(paste(num,"is not a prime number"))

}