Introduction to R

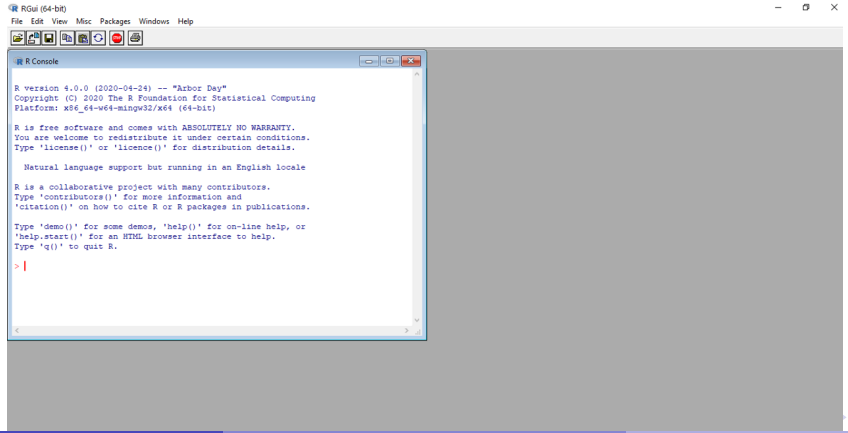
What is R?

It is an integrated suite of software facilitates for data manipulation, calculation and graphical display.
Among other things it has

- An effective data handling and storage facility.
- A suite of operators for calculations on array, in particular, matrices.
- A large, coherent, integrated collection of intermediate tools for data analysis.
- Graphical facilities for data analysis.
- A well developed, simple and effective programming language.

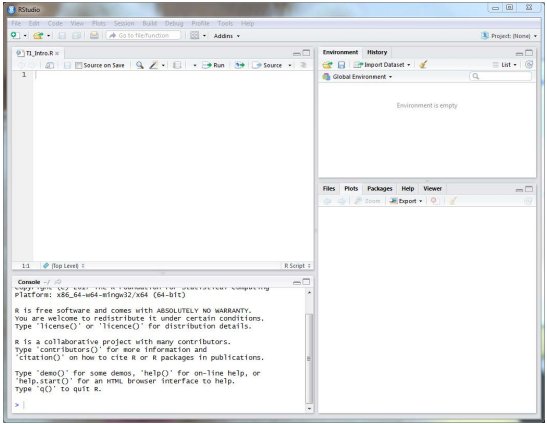
How To Start R?

- Downloading R from `https://www.r-project.org/`
- Double click the R's icon in the desktop to activate R (at least version 4.1.0 for our course).
- After R is started, R console is open in the RGui window.



RStudio

- A free integrated development environment (IDE) for R
- Have some features that makes it easier to work with
- Note: R still needs to be installed before RStudio



RStudio Suplements

DataCamp tutorial on “Working with the Rstudio IDE”

`https://www.datacamp.com/courses/working-with-therstudio-ide-part-1`

Working Directory

- The folder on your computer in which you are currently working.

- R will read and write files from/to this folder.

```
> setwd("~/Documents/BT1101")  # set working directory in Mac
> setwd("D:/BT1101")    # set working directory in Windows
> getwd()               # get current directory
```

- In RStudio, use dropdown menu to select working directory Session –> Set Working Directory –> Choose Directory
  Or
  Files Pane –> Navigating to a Directory –> Clicking “More” –> “Set as Working Directory”

How To Handle Data in R

Four most frequently used types of data objects:

- Vector: set of elements of the same mode (logical; numeric; character).
- Matrix: set of elements appearing in rows and columns, where the elements are of the same mode.
- Dataframe:
  -Similar to the Matrix object but columns can have different modes.

  -The rows contain different observations from your study or measurements from your experiment;

  -The columns contain the values of different variables which may be of different modes.
- List: generalization of a vector – represents a collection of data objects.

Creating a Vector in R: "c" function

- To create a vector, the simplest way is using the concatenation "c" function.

```
> #creating a vector of numbers:
> number<-c(2,4,6,8,10); number
[1]  2  4  6  8 10
> # creating a vector of strings/characters:
> string<-c("weight", "height", "gender"); string
[1] "weight" "height" "gender"
> #creating a Boolean vector (T/F):
> logic<- c(T, T, F, F, T); logic
[1]  TRUE  TRUE FALSE FALSE  TRUE
```

Appending item(s) to the existing vector:

- What is c(number,12,14)? A vector of numbers.
- What is c(string, 12,14)? A vector of strings where "12" and "14" are treated as strings.

Creating a Vector in R: "numeric" function

The "numeric" function creates a vector with all its elements being 0.

```
> number.2<-numeric(3)
> number.2
[1] 0 0 0
> c(number, number.2)
[1]  2  4  6  8 10  0  0  0
```

Creating a Vector in R: "rep" function

The "rep" function replicates elements of vectors.
rep(a,b): replicate the item *a* by *b* times.

```
> #rep(a,b): replicate the item a by b times.
> number.3<-rep(2,3)
> number.3
[1] 2 2 2
> number.3<-rep(c(1,2),3)
> number.3
[1] 1 2 1 2 1 2
> rep(string,2)
[1] "weight" "height" "gender" "weight" "height" "gender"
```

Creating a Vector in R: "seq" function

seq(from = a, to = b, by = c): from the number *a* to number *b*, create a sequence of numbers evenly spread by a distance of *c*.

```
> seq(from=2, to=10, by=2)
[1]  2  4  6  8 10
> seq(from=2, to=10, length = 5)
[1]  2  4  6  8 10
> 1:5
[1] 1 2 3 4 5
> 1:5*2
[1]  2  4  6  8 10
> seq(2,10,2)
[1]  2  4  6  8 10
> seq(10)
 [1]  1  2  3  4  5  6  7  8  9 10
```

Suplements

DataCamp tutorial about variables in R:
`https://campus.datacamp.com/courses/free-introduction-to-r/chapter-1-intro-to-basics-1?ex=3`

Creating a Matrix: "matrix" function

- *matrix(v,r,c)*: take the values from vector *v* to create a matrix with *r* rows and *c* columns.
- By default, matrix is filled by column.

```
> v <- c(1:6)
> m <- matrix(v, nrow=2, ncol=3); m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> # to fill the matrix by rows:
> m <- matrix(v, nrow=2, ncol=3, byrow=T); m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Creating a Matrix: "rbind" and "cbind" functions

- To bind a row (or many rows) onto a matrix, the command *rbind* can be used.

```
> a <- c(1,2,3,4)
> b <- c(5,6,7,8)
> ab_row <- rbind(a,b); ab_row
  [,1] [,2] [,3] [,4]
a    1    2    3    4
b    5    6    7    8
```

- To bind a column (or many columns) onto a matrix, the command *cbind* can be used.

```
> ab_col <- cbind(ab_row, c(9,10)); ab_col
  [,1] [,2] [,3] [,4] [,5]
a    1    2    3    4    9
b    5    6    7    8   10
```

List in R

- List: a generic vector containing other objects

```
> list.1 <- list(10.5, 20, TRUE, "Daisy")
> list.1
[[1]]
[1] 10.5

[[2]]
[1] 20

[[3]]
[1] TRUE

[[4]]
[1] "Daisy"
```

List in R

```
> x = c(2,4,6,8) # length 4
> y = c(T, F, T) # length 3
> list.2 = list(name1 = x, name2 = y) # assign names to list members
> list.2
$name1
[1] 2 4 6 8

$name2
[1]  TRUE FALSE  TRUE
> list.2[1] # reference by index
$name1
[1] 2 4 6 8
> list.2$name1 # reference by name
[1] 2 4 6 8
```

A Dataframe in R (1)

- Dataframe is a list of vectors of equal length.
- A dataframe has rows and columns:
  - The rows contain different **observations or measurements**;
  - The columns contain the values of different **variables**.
- **All the values of the same variable must go in the same column**.

Example: an experiment with three treatments (control, pre-heated and pre-chilled), and four measurements per treatment. A dataframe is created based on the given measurements.

Control	Pre-heated	Pre-chilled
6.1	6.3	7.1
5.9	6.2	8.2
5.8	5.8	7.3
5.4	6.3	6.9

This is a wrong dataframe.

A Dataframe in R (2)

The correct dataframe should be

Response	Treatment
6.1	Control
5.9	Control
5.8	Control
5.4	Control
6.3	Pre-heated
6.2	Pre-heated
5.8	Pre-heated
6.3	Pre-heated
7.1	Pre-chilled
8.2	Pre-chilled
7.3	Pre-chilled
6.9	Pre-chilled

- This has 2 variables: measurements as the response variable and another variable (called "treatment") for three levels of experimental factor.

Reading Data Files into R

There are several ways of reading/importing data files into R:

- *read.table(...)*
- *read.csv(...)* can be used to read dataframes from files using comma to separate values (.csv files). **This is the most commonly used in our course**.
- When reading from **an Excel file**, a simple method is to save each worksheet separately as a csv file and use *read.csv(....)* on each saved csv file.

Import a Free Format Data File (1)

- The first line contains the names of variables, then we use: *header* = *TRUE*.

```
> data1<-read.csv("C:/Data/crab.txt", sep = "", header = FALSE)
> data1[1:8,] #first 8 rows
     V1    V2    V3     V4     V5
1 color spine width satell weight
2     3     3  28.3      8  3.050
3     4     3  22.5      0  1.550
4     2     1  26.0      9  2.300
5     4     3  24.8      0  2.100
6     4     3  26.0      4  2.600
7     3     3  23.8      0  2.100
8     2     1  26.5      0  2.350
> names(data1)
[1] "V1" "V2" "V3" "V4" "V5"
```

Import a Free Format Data File (2)

- With *header* = *TRUE*.

```
> data1<-read.csv("C:/Data/crab.txt",sep = "", header = TRUE)
> data1[1:8,] #first 8 rows
  color spine width satell weight
1     3     3  28.3      8   3.05
2     4     3  22.5      0   1.55
3     2     1  26.0      9   2.30
4     4     3  24.8      0   2.10
5     4     3  26.0      4   2.60
6     3     3  23.8      0   2.10
7     2     1  26.5      0   2.35
8     4     2  24.7      0   1.90
> names(data1)
[1] "color"  "spine"  "width"  "satell" "weight"
```

Import a Free Format Data File (2)

- If the first line of the data file does not contain the names of the variables like the file `ex_1.txt`, we can create a vector to store the variable names:

```
> varnames <- c("Subject", "Gender", "CA1", "CA2", "HW")
> data2<-read.table("C:/Data/ex_1.txt", header = FALSE,
+                   col.names = varnames)
> data2
  Subject Gender CA1 CA2 HW
1      10      M  80  84  A
2       7      M  85  89  A
3       4      F  90  86  B
4      20      M  82  85  B
5      25      F  94  94  A
6      14      F  88  84  C
```

Missing values are denoted by NA.

Importing a Comma Separated Data

We can use *read.csv* function:

```
> data3<-read.csv("C:/Data/ex_1_comma.txt",sep = ",", header = FALSE)
```

Note that file `ex_1_comma.txt` does not have names of columns, and the values are separated by a comma.

Assessing Parts of a Dataframe

- Read data into R:

```
> data3<-read.table("C:/Data/ex_1_name.txt", header = TRUE)
> data3
  Subject Gender CA1 CA2 HW
1      10      M  80  84  A
2       7      M  85  89  A
3       4      F  90  86  B
4      20      M  82  85  B
5      25      F  94  94  A
6      14      F  88  84  C
```

- Get a specific variable/column by the column name

```
> attach(data3)
> CA1
[1] 80 85 90 82 94 88
```

Assessing Parts of a Dataframe: Logical Tests

- Selecting some specified variables (columns):

```
> data3[,1]
[1] 10  7  4 20 25 14
> data3[,2:4]
  Gender CA1 CA2
1      M  80  84
2      M  85  89
3      F  90  86
4      M  82  85
5      F  94  94
6      F  88  84
```

- Selecting some specified observations (rows):

```
> data3[1:2,] # row 1 to row 2
  Subject Gender CA1 CA2 HW
1      10      M  80  84  A
2       7      M  85  89  A
```

Assessing Parts of a Dataframe: Logical Tests

- Selecting some specific values in the data

```
> data3[3,3] # value at 3rd row & 3rd column
[1] 90
> data3[3,4] # value at 3rd row & 4th column
[1] 86
```

Assessing Parts of a Dataframe: Logical Tests

- Select all observations/rows by some conditions

```
> # all the rows (observations) whose gender = M:
> data3[Gender == "M",]
  Subject Gender CA1 CA2 HW
1      10      M  80  84  A
2       7      M  85  89  A
4      20      M  82  85  B
> #all the rows (observations) whose gender = M and CA2>85
> data3[Gender == "M" & CA2 > 85,]
  Subject Gender CA1 CA2 HW
2       7      M  85  89  A
```

Common Commands

x and y are vectors. Some functions on vectors in R:

- max(x): maximum value of vector x
- min(x): minimum value of x
- sum(x): total of all the values in x
- mean(x): arithmetic average values in x
- range(x): min(x) and max(x)
- cor(x,y): correlation value between vectors x and y
- sort(x): a sorted version of x

Common Commands Used for a Dataframe

- Read/import a dataframe into R: data = read.csv("crab.txt"...)
- names(data): to get the names of columns in `data`
- attach(data)
- colMeans(data): get the mean of every column, if all columns are numeric
- which(data$x1 == 3): get the index of all the rows of "data" that the column **x1** has value 3.

Common Plots

Chart Type	R Functions
Pie Chart	pie(x, labels, radius, main, col, clockwise)
Bar Chart	barplot(H, xlab, ylab, main, names.arg, col)
Box Chart	boxplot(x, data, notch, varwidth, names, main)
Histogram	hist(v,main,xlab,xlim,ylim,breaks,col,border)
Line Graph	plot(v,type,col,xlab,ylab)
Scatterplots	plot(x, y, main, xlab, ylab, xlim, ylim, axes)

- **while** loop
- **for** loop
- Conditioning with **if...else**
- Define a function

while Loop: Examples

- A simple `while` loop

```
> x = 1
> while(x<=3) {print("x is less than 4")
+               x = x+1}
[1] "x is less than 4"
[1] "x is less than 4"
[1] "x is less than 4"
```

- Find the sum of first 10 integers:

```
> x<-0; S<-0
> while(x<=10) {S<- S+ x
+                x<-x+1}
> S
[1] 55
```

while Loop

- The `while` loop is in the form of
  `while (condition) {expression}`

- How this loop works:
  1. `(condition)` must evaluate to a TRUE or a FALSE.
  2. If `(condition)` is TRUE, do all the steps inside the code block of `{expression}`.
  3. Check `(condition)` again
  4. Repeat [2] and [3] above until `(condition)` is a FALSE.

for Loop

- Example: find the sum of first 10 integers

```
> S<-0; for(i in 1:10){S <-S+i}
> S
[1] 55
```

- Find the mean

```
> x = c(2, 4, 3, 8, 10)
> l = length(x)
> S = 0
> for (i in 1:l){S = S + x[i]}
> ave = S/l; ave
[1] 5.4
```

for Loop

- The for loop is in the form of:
  `for (<variable> in <range>) {expression}`

- How this loop works:

  Each time through the loop, <variable> takes a value

  1. First time, <variable> starts at the smallest value in the range and do all the steps inside the {expression}.

  2. Next time, <variable> gets the previous value + 1, and do all the steps inside the {expression}, until <variable> reaches the last value in the range.

Conditions

- Find the sum of all even numbers from 1 up to 100.

```
> x = c(1:100); S = 0
> for (i in 1:length(x)){
+   if(x[i]%%2 ==0){S = S + x[i]} else {S = S}
+ }; print(S)
[1] 2550
```

if...else

```
> x = c(1:10);
> S = numeric(0)
> M = numeric(0)
> L = numeric(0)
> for (i in 1:length(x)){
+   if (x[i] <=3){S = append(S, x[i])} else if (x[i]< 8)
+     {M = append(M, x[i])} else {L = append(L, x[i])}
+ }
> print(S)
[1] 1 2 3
> print(M)
[1] 4 5 6 7
> print(L)
[1]  8  9 10
```

ifelse

```
> x = c(1:8);x
[1] 1 2 3 4 5 6 7 8
> y = ifelse(x%%2 == 0, "even", "odd")
> y
[1] "odd"  "even" "odd"  "even" "odd"  "even" "odd"  "even"
```

Functions

- Defining functions will be introduced in Topic 2 along with the basic probability and statistics.
- A good source for reading details about building functions in R: `https://datasciencebeginners.com/2018/11/02/10-user-defined-functions-in-r/`
- Tutorial about functions on DataCamp