Introduction to R

1. Introduction
2. Vectors in R
3. Matrices in R
4. Lists in R
5. Dataframes in R
6. Commonly Used Commands
7. Loops, Conditions and Functions in R

1. Introduction
2. Vectors in R
3. Matrices in R
4. Lists in R
5. Dataframes in R
6. Commonly Used Commands
7. Loops, Conditions and Functions in R

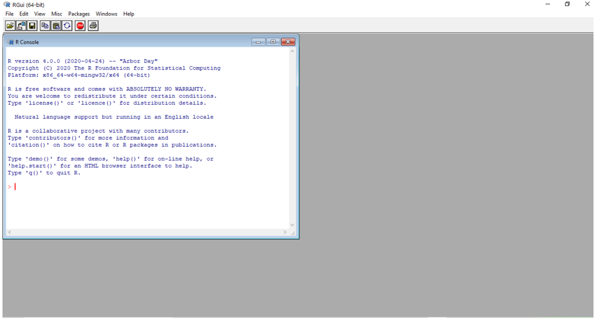
What is R?

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?

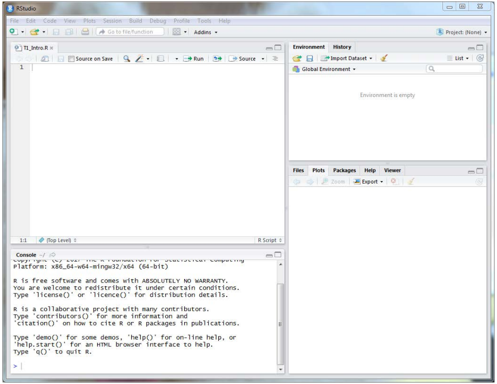
- Downloading R from `https://www.r-project.org/`
- Double click the R's icon in the desktop to activate R (at least version 4.1.0 for our course).
- After R is started, R console is open in the RGui window.



```
R version 4.0.0 (2020-04-24) -- "Arbor Day"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

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



Environment is empty

```
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

RStudio Suplements

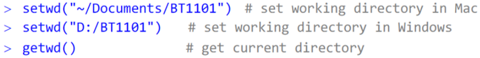
DataCamp tutorial on "Working with the Rstudio IDE"

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 drop down 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.

1. Introduction
2. Vectors in R
3. Matrices in R
4. Lists in R
5. Dataframes in R
6. Commonly Used Commands
7. Loops, Conditions and Functions in R

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`

1. Introduction
2. Vectors in R
3. Matrices in R
4. Lists in R
5. Dataframes in R
6. Commonly Used Commands
7. Loops, Conditions and Functions in R

Creating a Matrix: “matrix()” function

- Members in a matrix must be of the same mode (numeric or character).
- $matrix(v, nrow = r, ncol = c)$: 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) # a vector with 6 elements
> m <- matrix(v, nrow=2, ncol=3)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Creating a Matrix: “matrix()” function

To fill the matrix by row instead of column, we need to specify `byrow = TRUE`.

```
> # 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()” 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)
> # a and b are two vectors of the same length
> ab_row <- rbind(a,b)
> ab_row # each vector is one row of the matrix
  [,1] [,2] [,3] [,4]
a    1    2    3    4
b    5    6    7    8
```

Creating a Matrix: “cbind()” functions

```
> a <- c(1,2,3,4)
> b <- c(5,6,7,8)
```

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

```
> ab_col <- cbind(a,b)
> ab_col # each vector is one column of the matrix
     a b
[1,] 1 5
[2,] 2 6
[3,] 3 7
[4,] 4 8
```

1. Introduction
2. Vectors in R
3. Matrices in R
4. Lists in R
5. Dataframes in R
6. Commonly Used Commands
7. Loops, Conditions and Functions in R

List in R

- Lists are the objects which contain elements of **different modes** like numbers, strings, vectors and another list inside it.

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

[[2]]
[1] TRUE

[[3]]
[1] "Daisy"
```

List in R

```
> x = c(2,4,6,8) # vector length 4, numeric
> y = c(T, F, T) # vector length 3, logical
> list.2 = list(A = x, B = y)
> # Create a list using two vectors, x and y.
> # and assign names A, B to the members of the list
> list.2
$A
[1] 2 4 6 8

$B
[1]  TRUE FALSE  TRUE
```

List in R

- Refer to item in a list by index:

```
> list.2[1] # get the first item in the list
$A
[1] 2 4 6 8
```

- Refer to an item by the name of the item:

```
> list.2$A # get item A (a vector) in the list
[1] 2 4 6 8
```

- Refer to a member in an item of the list:

```
> list.2$A[1]
[1] 2
```

List in R

-
  ```
  > list.1[1] # this is an item of list, NOT numeric
  [[1]]
  [1] 10.5
  ```

-
  ```
  > list.1[[1]] # this is numeric
  [1] 10.5
  ```

1. Introduction
2. Vectors in R
3. Matrices in R
4. Lists in R
5. Dataframes in R
6. Commonly Used Commands
7. Loops, Conditions and Functions in R

A Dataframe in R

- Dataframe is a list of vectors of equal length.

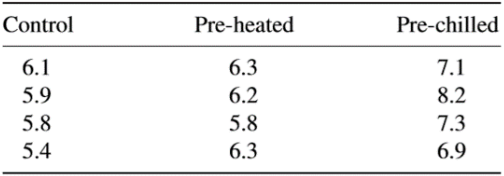
- A dataframe has rows and columns:
  - The rows contain different **observations or measurements**;
  - The columns contain the values of different **variables**.

- **All the values of the same variable must go in the same column.**

Notes

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

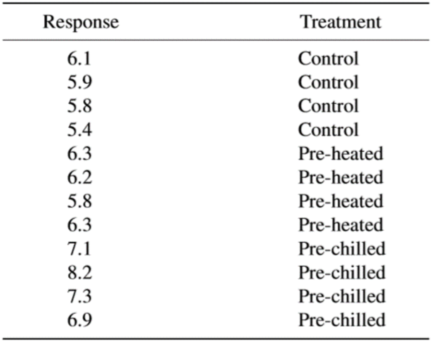
- Is it a correct dataframe in R?



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

A Dataframe in R

The correct dataframe for the example in the previous slide 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

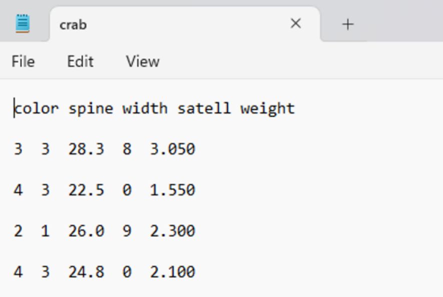
The very first step is to make sure that the **file is under the working directory that R is accessible to**.

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

- *read.table(file, header = FALSE, sep = "")*
- *read.csv(file, header = TRUE, sep = ",")*

Open the file in Notepad

- When you have a file, it's good to open that file in Notepad **before importing to R** to know if the file has header; if the values are separated by comma or by space, or by tab, etc.
- Open the file `crab.txt`, we can observe that it has header, and the values are separated by space.



```
color spine width satell weight
3  3  28.3  8  3.050
4  3  22.5  0  1.550
2  1  26.0  9  2.300
4  3  24.8  0  2.100
```

Import a File into R

- Since the file has header, then we use: *header* = *TRUE*; The values are separated by space, hence we use `sep = ""`.

```
> data1<-read.csv("C:/Data/crab.txt", sep = "", header = TRUE)
> # If you have set the working directory
> # setwd(C:/Data)
> # then you just need to have the code as
> # data1<-read.csv("crab.txt", sep = "", header = TRUE)
```

Import a File into R

- How the dataframe looks like:

```
> data1[1:4,] #first 4 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
> names(data1) # names of the columns
[1] "color"  "spine"  "width"  "satell" "weight"
```

- `names(data1)` returns a vector with 5 elements that are the names of 5 columns.

Import a Data File

- If the first line of the data file does not contain the names of the variables like the file ex_1.txt.

```
> data2<-read.table("C:/Data/ex_1.txt", header = FALSE)
```

- This is how data2 looks like where the columns are named by R.

```
> data2
  V1 V2 V3 V4 V5
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
```

Changing Names of Columns

```
> names(data2)
[1] "V1" "V2" "V3" "V4" "V5"
```

- We can change the names of the columns

```
> names(data2) = c("Subject", "Gender", "CA1", "CA2", "HW")
> 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
```

Assessing Parts of a Dataframe

NAME[x, y]

- `NAME` is the name of the dataframe.
- `x` is to indicate the rows that we want to select. If it's not specify, then all rows are selected.
- `y` is to indicate the columns that we want to select. If it's not specify, then all columns are selected.
- `x` could be a number; a vector of numbers; or a condition; Similar for `y`.

Assessing Parts of a Dataframe

- The 2nd rows of a dataframe and all columns are listed.

```
> data2[2,] # 2nd rows
  Subject Gender CA1 CA2 HW
2       7      M  85  89  A
```

- Third column

```
> data2[,3]
[1] 80 85 90 82 94 88
```

- Get a specific variable/column by the column name

```
> data2$CA1 # assess a column by its name
[1] 80 85 90 82 94 88
```

Command attach()

- Command attach(data2) could help us to assess every column of data2 by the name of the column.

- However, the properties of each column inside data2 will be kept as when it was "attached".

```
> attach(data2)
> CA1
[1] 80 85 90 82 94 88
```

Assessing Parts of a Dataframe

- Selecting a specified column:

```
> data2[,1] # select only the first column
[1] 10  7  4 20 25 14
```

- Select a few columns:

```
> data2[,2:4] # select columns from 2 to 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
```

Assessing Parts of a Dataframe

- Selecting some specified observations (rows):

```
> data2[1:2,] # select 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

- Selecting some specific values in the data

```
> data2[3,3] # value at 3rd row & 3rd column
[1] 90
> data2[3,4] # value at 3rd row & 4th column
[1] 86
```

Assessing Parts of a Dataframe by Comditions

- Select all observations that are males:

```
> data2[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
```

- Select all observations that are males and their CA2 is more than 85:

```
> data2[Gender == "M" & CA2 > 85,]
  Subject Gender CA1 CA2 HW
2       7      M  85  89  A
```

1. Introduction
2. Vectors in R
3. Matrices in R
4. Lists in R
5. Dataframes in R
6. Commonly Used Commands
7. Loops, Conditions and Functions in R

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

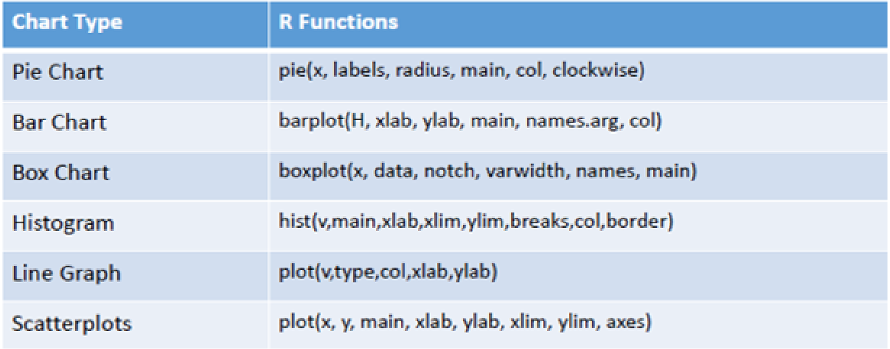


Chart Type	R Functions
Pie Chart	pie(x, labels, radius, main, col, clockwise)
Bar Chart	barplot(H, xlab, ylab, main, names.arg, col)
Box Chart	boxplot(x, data, notch, varwidth, names, main)
Histogram	hist(v,main,xlab,xlim,ylim,breaks,col,border)
Line Graph	plot(v,type,col,xlab,ylab)
Scatterplots	plot(x,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

1. Introduction
2. Vectors in R
3. Matrices in R
4. Lists in R
5. Dataframes in R
6. Commonly Used Commands
7. Loops, Conditions and Functions in R

- **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.

if...else

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

```
> x = c(1:100); S = 0
```

- We'll identify all the even numbers in vector `x` and sum them up. For each element `x[i]` of `x`, we'll check the remainder when it divides by 2; if the remainder is 0, then we add that element `x[i]` to the sum `S`; if the remainder is not 0, we do nothing to the sum.

```
> 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);
```

Vector x has 10 numbers from 1 to 10. We now need to create 3 vectors, S, M, L where S contains all the values in x that are at most 3; M contains all the values in x that are less than 8 but more than 3; L contains all the rest.

```
> # STEP 1: create 3 vectors S, M, L which has nothing yet
> S = numeric(0)
> M = numeric(0)
> L = numeric(0)
```

if...else

STEP 2: checking each member of `x[i]`; then to append it to `S`, or `M` or `L`:

```
> 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
```

We need to replace each number in `x` by a word such that: if the number is odd, we replace that number by the word “odd”; if the number is even, we then replace it by the word “even”.

It could be done by the command `ifelse()`

```
> x = ifelse(x%%2 == 0, "even", "odd")
> x
[1] "odd"  "even" "odd"  "even" "odd"  "even" "odd"  "even"
```

User-Definied Functions in R

- Characteristics of a function:
  -has a name
  -has parameters
  -has a body
  -returns something

- How to write/define:
  `name <- function(parameters) { function body }`

- At the end of the `function body`, some command to ask for evaluation and return normally be included.

Function to calculate OR

- We would want to form a function that helps us to calculate OR for a 2 × 2 matrix or table.

```
> OR<-function(x){
+   if(any(x==0)) {x<-x+0.5}
+   odds.ratio<-x[1,1]*x[2,2]/(x[2,1]*x[1,2])
+
+   return(odds.ratio) }
> #OR(table)
>
```

- Function is named as “OR”.
- It has one argument/parameter, `x`, which is a 2 × 2 matrix/table.
- Question: write a function to find the median of a given vector.