Introduction to R?

What is R?

● R is a comprehensive statistical environment and programming language for professional data analysis and graphical display.

● Webpage: http://www.r-project.org

Key Advantages:

● R is free

● New statistical methods are usually first implemented in R

● Lots of help due to active community

R Studio

● Powerful IDE (Integrated Development Environment) for R

● It is free and open source, and works on Windows, Mac, and Linux and over the web

● Webpage: https://www.rstudio.com/

How R works ?

R commands are organized in packages (also called libraries)
Examples: stats, datasets, ggplot2, dplyr
To use a package, it has to be installed AND loaded!

Which packages are loaded at start?
library(lib.loc=.Library)

Which packages are installed?
installed.packages()

How to install packages ??
install.packages("packagename")
Load package: library(package name)

How to get help??
library(help="package")
??package

Using R : Best Practices

How to organize a R session :

● Open RStudio or a R console

● Open a new or pre-existing script in the text editor or RStudio (extension .R)

● Save the file (for example as ‘Day1.R’)

● Set your working directory (wd) with setwd(“path2directory)

● Check your working directory with getwd()

● Load (and install) required packages – Install with install.packages(“name”) - only once, need to specify CRAN mirror – Load with library(name) – each session if required

● Comment your script with # – REALLY IMPORTANT!

● Write and execute your commands (with button or 'Ctrl+Enter' in Rstudio)

● Output is saved in your working directory (if folder unspecified)

● Save your script ('Ctrl+S’)

● Quit your session and save workspace if required (q() in console)

R : Just a calculator

R understands the following basic operators:

1. + and − for addition and subtraction
2. ∗ and / for multiplication and division
3. ∧ for exponents
4. %% is the modulo operator
5. %\% for integer division

Try yourself:

```
exp(1); exp(log(5))
sin(pi/2)
cos(pi/2)
max(4,2,5,1); min(4,2,5,1)
sum(4,2,5,1); prod(4,2,5,1)
sqrt(16)
factorial(4)
choose(5,2)
```

R : Object Oriented Programming

Everything in R is an object

- An object is a data structure having some attributes and methods which act on its attributes.
- Class is a blueprint for the object. We can think of class like a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house.
- House is the object. As, many houses can be made from a description, we can create many objects from a class. An object is also called an instance of a class and the process of creating this object is called instantiation.

Variables : Building block for R

Variables are reserved memory locations to store values. This means that, when you create a variable you reserve some space in memory.

Data types

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

- logical
- numeric
- integer
- character
- complex

Data Structures

The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable.

- vector
- factor
- list
- matrix
- dataframe

Data types

Data type	Description	Example
Logical	Binary	True or False
Numeric	Integers and real numbers	5, -2, 3.1415, sqrt(2)
Integer	Whole numbers	5L,6L
Character	Character String	“I love India”
Complex	Complex numbers	2+3i

- Types can be explicitly converted: as.logical(), as.integer(), as.numeric(), as.complex(), as.character()
- You can check for a data type: is.logical(), is.integer(), is.numeric(), is.complex(), is.character()

Data Structures

Vectors

What is a vector?

- A vector is a collection of values that all have the same data type
- One-dimensional

Examples:

- (-2, 3.4, 3.75, 5.2, 6)
- (TRUE, FALSE, TRUE, TRUE, FALSE)
- (“blue”, “green”, “red”, “red”)

You can create a vector with different functions:

- **c()** function to combine individual values
- **seq()** to create more complex sequences
- **rep()** to create replicates of values

Type conversion

It is important to remember that a vector can only be composed of one data type. This means that you cannot have both a numeric and a character in the same vector. If you attempt to do this, the lower ranking type will be *coerced* into the higher ranking type.

logical < integer < numeric < complex < character

Vectors : Useful Functions

Function	Description
sum()	Sum of elements
prod()	Product of elements
Min()	Minimum value
Max()	Maximum value
Mean()	Mean value
Median()	Median value
Which()	Index after evaluating logical expression
Unique()	Unique element list
Range()	Range
Sd()	Standard deviation
Sort()	Sort - Decreasing - by default
Length()	No of elements
Summary()	Summary statistics

Factors

What is a factor?

- A factor is used to store categorical data
- Can only contain predefined categories or levels
- Can be ordered and unordered

Examples:

- ("yes", "no", "yes", "yes")
- ("male", "female", "female", "male")
- ("small", "large", "small", "medium")

Helpful commands :

- Factors can be created using **factor()**
- The levels of a factor can be displayed using **levels()**

Lists

What is a List?

- A collection of data structures
- A list can encompass any data types, including lists
- Objects can have different lengths
- Almost all functions (e.g., t-test, linear regression, etc.) in R produce output that is stored in a list

Examples:

- List < list(1:3, c("a", "b"), c(TRUE, FALSE, TRUE))

Helpful commands :

- You can construct lists by using **list()**

Matrices

What is a Matrix?

• In mathematics, a matrix (plural matrices) is a rectangular array of numbers, symbols, or expressions arranged in rows and columns.

• The individual items in a matrix are called its elements or entries.

Examples:

$$A = \begin{bmatrix} -5 & 1 & -3 \\ 6 & 0 & 2 \\ 2 & 6 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 4 & 5 \\ -8 & 10 & 3 \\ -2 & -3 & -9 \end{bmatrix}$$

$$A + B = \begin{bmatrix} -3 & 5 & 2 \\ -2 & 10 & 5 \\ 0 & 3 & -8 \end{bmatrix} \qquad A - B = \begin{bmatrix} -7 & -3 & -8 \\ 14 & -10 & -1 \\ 4 & 9 & 10 \end{bmatrix}$$

How matrices can be created :

- 1. matrix() - Function
- 2. Converting vector into matrix
- 3. Binding together vectors

Data Frames

What is a Data frame?

- A collection of vectors that are of equal length
- Two-dimensional, arranged in rows and columns
- Columns can contain vectors of different data types

BUT : *WITHIN a column, every cell must be the same type of data!*

- Used to represent entire data sets

Data frames can be created using the function:

- data.frame() - creates a data frame object from a set of vectors

Indexing

Indexing by an integer vector :

- You can use x[ ] to look up a single element or multiple elements
- You can also use negative integers to return a vector consisting of all elements except the specified elements
- In multidimensional data structures (e.g. matrices and data frames) an element at the mth row, nth column can be accessed by the expression x[m, n]
- The entire m-th row can be extracted by the expression x[m,]
- The entire n-th column can be extracted by the expression x[, n]
- Multiple rows and columns can also be extracted

Indexing by name :

- You can index an element by name using the $ notation
- You can also use the single-bracket notation [ ] to index a set of elements b name

Practice Questions : Vectors

Set 1

- Define the variable v1 as the vector (3, 7, −4, 0)
- Define the variable v2 as the vector (1, 2, 3, . . . , 48, 49, 50)
- Define the variable v3 as the vector (3, 7, −4, 0, 1, 2, 3, . . . , 48, 49, 50)
- Define the variable v4 as the vector (0.0, 0.1, 0.2, 0.3, . . . , 1.8, 1.9, 2.0)
- Sum over all elements of v1. Sum over all elements of v2.
- What is the product of all elements of the vector (10, 11, 12, 13, . . . , 19, 20)?

Set 2

- Define the vector data as data <- 90*1:100 - (1:100)^2 + 1000
- What is the length of the vector data?
- What is the first, the seventeenth and the last entry of the vector data?
- What is the maximum of the vector data? At which index is the maximum attained?
- Plot the vector data with plot(data) and visually confirm your last result.
- At which indices are the entries of data between 2000 and 2500?
- Define a vector half that contains only the last half of the elements of data. Use negative integers to perform this tasks.

Practice Questions : Lists and Matrices

Lists

- Define the list myList as myList <- list(1:6, c("a", "b"), c(FALSE, TRUE, TRUE)
- What is the element with index 2 in myList?
- Which type of data is the element with index 3 in myList

Matrices

- Create the following matrices

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 2 & 3 & 4 \\ 7 & 8 & 9 \end{pmatrix}$$

- Define a new matrix m by m <- matrix(11:35, nrow=5, byrow=TRUE)
- What is the entry in the third row and forth column?
- Define a new submatrix sub that contains the elements of rows 2 to 4 and columns 3 to 5.
- Assign the names 'Variable1', 'Variable2' and 'Variable3' to the columns of sub.

Practice Questions : Data frames

- Use the command **data.frame()** to create a data frame with the following entries

name	degree	grade
Leonie	Bachelor	2.3
Luca	Master	3.0
Leon	Bachelor	2.0
Lea	Bachelor	1.3
Luis	Master	2.7
Laura	Master	1.0

- Get an overview of results with the commands names(), str() and summary().
- Use the $ operator to extract the column ’grade’ from ’results’.
- Which command returns the fifth element of the vector ’grade’?
- Create a new data frame students that contains only the vectors ’name’ and ’degree’. Do not use the command data.frame() for this task.
- We wish to change ’degree’ into ’deg’ to save typing work. Use the command names() to accomplish this change. You might need to consult the help page ?names to find out how to do this

Reading and Writing data

Basic workflow :

1) Import your data
2) Check, clean and prepare your data (can be up to 80% of your project)
3) Conduct your analyses
4) Export your results
5) Clean R environment and close session

Basic Sanity check for data :

- Columns should contain variables
- Rows should contain observations, measurements, cases, etc.
- Use first row for the names of the variables
- Enter NA (in capitals) into cells representing missing values
- You should avoid names (or fields or values) that contain spaces
- Store data as .csv or .txt files as those can be easily read into R

Example

Data of 3 groups/treatments: Control, Tropics, Temperate - 4 measurements per treatment

Control	Tropics	Temperate
6.1	6.3	7.1
5.9	6.2	8.2
5.8	5.8	7.3
5.4	6.3	6.9

Response	Group
6.1	Control
5.9	Control
5.8	Control
5.4	Control
6.3	Tropics
6.2	Tropics
5.8	Tropics
6.3	Tropics
7.1	Temperate
8.2	Temperate
7.3	Temperate
6.9	Temperate

One of these is not a data frame. Which one ??

Import data

- Import data using read.table() and read.csv() functions

Example :

Data <- read.table(file = "datafile.txt")

Data <- read.csv(file = "datafile.csv")

Creates a data frame named Data

Error in file(file, "rt") : cannot open the connection
In addition: Warning message: In file(file, "rt") :
cannot open file 'datafile.csv': No such file or directory

Note - ***Set your working directory (setwd()) first, so that R uses the right folder to look for your data file! And check for typos!***

To be continued ..

Import data - *Continued..*

Useful Arguments –

- The header = TRUE argument tells R that the first row of your file contains the variable names
- The sep = ”," argument tells R that fields are separated by comma
- The strip.white = TRUE argument removes white space before or after factors that has been mistakenly inserted during data entry (e.g. “small” vs. “small ” become both “small”)
- The na.strings = " " argument replaces empty cells by NA (missing data in R)

Syntax :

```
Data <-read.csv(file = "datafile.csv”, header = TRUE, sep = ”,”, strip.white = TRUE, na.strings = " ")
```

Data cleaning

- Import the sample data into a variable Snail_data

Snail_data < read.csv(file = "Snail_feeding.csv", header = TRUE, strip.white = TRUE, na.strings = " ")

- Use the str() command to check the status and data type of each variable:

str(Snail_data)

```
'data.frame':   769 obs. of  10 variables:
 $ Snail.ID: int  1 1 1 1 1 1 1 1 1 1 ...
 $ Sex     : Factor w/ 5 levels "female","male",..: 2 2 5 2 2 2 2 2 2 2 ...
 $ Size    : Factor w/ 2 levels "large","small": 2 2 2 2 2 2 2 2 2 2 ...
 $ Feeding : logi  FALSE FALSE FALSE FALSE FALSE TRUE ...
 $ Distance: num  0.17 0.87 0.22 0.13 0.36 0.84 0.69 0.6 0.85 0.59 ...
 $ Depth   : num  1.66 1.26 1.43 1.46 1.21 1.56 1.62 162 1.96 1.93 ...
 $ Temp    : int  21 21 18 19 21 21 20 20 19 19 ...
 $ X       : logi  NA NA NA NA NA NA ...
 $ X.1     : logi  NA NA NA NA NA NA ...
 $ X.2     : logi  NA NA NA NA NA NA ...
```

To get rid of the extra columns we can just choose the columns we need by using Snail_data[m, n]

we are interested in columns 1:7

Snail_data < Snail_data[, 1:7]

get an overview of your data

str(Snail_data)

Data cleaning

Something seems to be weird with the column 'Sex'

unique(Snail_data$Sex)

Or

levels(Snail_data$Sex)

To turn “males” or “Male” into the correct “male”, you can use the [ ]-Operator together with the which() function:

Snail_data$Sex[which(Snail_data$Sex == "males")] <- "male”

Snail_data$Sex[which(Snail_data$Sex == "Male")] <- "male”

Or both together:

Snail_data$Sex[which(Snail_data$Sex == "males" | Snail_data$Sex == "Male")] <- "male“

Check if it worked with unique()

unique(Snail_data$Sex)

- The summary() function provides summary statistics for each variable:

Overview Statistics

- After you read in your data, you can briefly check it with some useful commands:

- summary() provides summary statistics for each variable
- names() returns the column names
- str() gives overall structure of your data
- head() returns the first lines (default: 6) of the file and the header
- tail() returns the last lines of the file and the header

Now we will look at a few common things to look out for !!

Adding variables to a data frame

There are three ways to do this

- Using $
  - Snail_data$log_Depth <- log(Snail_data$Depth)
- Using the [ ] - operator
  - Snail_data[, "log_Depth"] <- log(Snail_data$Depth)
- Using the function mutate() from dplyr package
  - Snail_data <- mutate(Snail_data, log_Depth = log(Depth))

Duplication in data

- Function: duplicated()

Example:

duplicated(Snail_data)

- You should check how many of these rows are duplicate entries

sum(duplicated(Snail_data))

Think: Why does it actually work with sum()

- You probably want to know WHICH row is duplicated:

Snail_data[which(duplicated(Snail_data)),]

Duplicate rows are removed so as to not skew the analysis

Missing values in data

Name	Weight	Gender	Play Cricket/ Not
Mr. Amit	58	M	Y
Mr. Anil	61	M	Y
Miss Swati	58	F	N
Miss Richa	55		Y
Mr. Steve	55	M	N
Miss Reena	64	F	Y
Miss Rashmi	57		Y
Mr. Kunal	57	M	N

Gender	#Students	#Play Cricket	%Play Cricket
F	2	1	50%
M	4	2	50%
Missing	2	2	100%

Name	Weight	Gender	Play Cricket/ Not
Mr. Amit	58	M	Y
Mr. Anil	61	M	Y
Miss Swati	58	F	N
Miss Richa	55	F	Y
Mr. Steve	55	M	N
Miss Reena	64	F	Y
Miss Rashmi	57	F	Y
Mr. Kunal	57	M	N

Gender	#Students	#Play Cricket	%Play Cricket
F	4	3	75%
M	4	2	50%

Why do we care about missing values ??

What are the methods used to treat missing values ?

1) Deletion – Missing values are random in nature and effect a statistically insignificant fraction of the sample
2) Mean/Mode/Median imputation
    1) Generalized imputation
    2) Similar case imputation
3) Prediction model – Clustering techniques

- Important command: is.na()

Example :

v < c(1, 3, NA, 5)

is.na(v)

- Ignore missing data:

na.rm=TRUE

Example :

mean(v)

mean(v, na.rm=TRUE)

Practice Questions

Exercise 1: Get an overview of the data.

- Import the sparrow data using read.table()
- Get an overview of the sparrow data with the command str().
- Return the minimum, median, mean and maximum for tarsus and bill measurements.

  Hint: You may use a single function to perform this task.

Exercise 2: Checking and cleaning data frames.

- During the data entry, three rows have been entered twice. Which are these duplicate rows? Remove the duplicate rows from the data frame. Hint: It might be faster if you incorporate the function which().
- Display the levels of the factor Sex. Correct the typos by using which() and logical operators, such that Sex contains only the levels ’Male’ and ’Female’. Remove all other extra levels.

Exercise 3: Missing values.

- Find out which rows in the variable Wing contains NAs.
- Replace all NAs with the values 59, 56.5, and 57 (in this order). Use which(is.na()) to check if your replacement worked.

To be continued ..

Variable Manipulation : Dplyr

The package contains a set of functions (or “verbs”) that perform common data manipulation operations such as filtering for rows, selecting specific columns, re-ordering rows, adding new columns and summarizing data.

Some of the most important functions in the package are

1. select() – select columns
2. filter() – filter rows
3. arrange() – arrange rows
4. mutate() – add columns
5. summarise() – summarise values
6. group_by() – Allows groups operations

Dplyr - continued

Filter

```
filter(airquality, Temp > 70)
filter(airquality, Temp > 80 & Month > 5)
```

Mutate

```
mutate(airquality, TempInC = (Temp - 32) * 5 / 9)
```

Summarize

```
summarise(airquality, mean(Temp, na.rm = TRUE))
```

Group by

```
summarise(group_by(airquality, Month), mean(Temp, na.rm = TRUE))
```

Sample

```
sample_n(airquality, size = 10)
sample_frac(airquality, size = 0.1)
```

Arrange

```
arrange(airquality, desc(Month), Day)
```

Dplyr - continued

Pipe operator

The pipe operator in R, represented by %>% can be used to chain code together. It is very useful when you are performing several operations on data, and don't want to save the output at each intermediate step.

Let's say we want to remove all the data corresponding to Month = 5, group the data by month, and then find the mean of the temperature each month. There are two ways in which this can be done.

```
filteredData <- filter(airquality, Month != 5)

groupedData <- group_by(filteredData, Month)

summarise(groupedData, mean(Temp, na.rm = TRUE))
```

Or

```
airquality %>%
    filter(Month != 5) %>%
    group_by(Month) %>%
    summarise(mean(Temp, na.rm = TRUE))
```

Visualization - Graphics package

Simple graphics using plotting functions in the graphics package

- Base R, installed by default
- Easy and quick to type
- Wide variety of functions

We will look at three important charts

- Histogram – Frequency distribution
- Scatter plots – Relationship between continuous variables
- Boxplot – Relationship between continuous and categorical variables

Histograms - hist()

Univariate frequency spread of a variable

```
hist(Sparrows$Tarsus)
```

Greater customization is possible !!

```
hist(Sparrows$Tarsus, col = "grey", breaks = 50)
```

```
hist(Sparrows[Sparrows$Sex == "Male",]$Tarsus, col = "grey", breaks = 50)
```

Scatter plots - plot()

plot(Sparrows$Wing, Sparrows$Tarsus)

You can also alter axis limits and shape of symbols

plot(Sparrows$Tarsus, Sparrows$Wing, xlim = c(50, 70), pch = 15, col = “blue”)

What is wrong here ??

You can also alter the size of plotting symbols

plot(Sparrows$Wing, Sparrows$Tarsus, xlim = c(50,70), cex = 1.5)

Boxplots - boxplot()

```
boxplot(Wing ~ Sex, data = Sparrows)
```

Lets have a look at the important arguments

```
boxplot(Wing ~ Sex, data = Sparrows,
xlab = 'Sex',                                   # Adds label to x-axis
ylab = 'Wing length (mm)’,                      # Adds label to y-axis
col=c("red", "blue"),                           # Adds color
ylim = c(50,70),                                # Changes axis limits
main = "Boxplot”))                              # Adds title
```

Grouping with multiple variables is also possible

```
boxplot(Wing ~ Sex + Species, data = Sparrows, xlab = ’Species and Sex’,
ylab = 'Wing length (mm)', col=c("red", "blue"), ylim = c(50,70), main = ""))
```

Conditional Statements

if(), else() and ifelse()

Syntax:

if (condition) { commands1 }

if (condition) { commands1 } else { commands2 }

ifelse(conditions vector, yes vector, no vector)

These are conditional statements that are executed based on one or multiple conditions.

Example:

x <- 4

if (x==5) {x <- x+1} else {x <- x*2}   What is the response ??

y <- 1:10

z <- ifelse(y<6, y^2, y-1)   What is the response ??

Loops

for(), while() and repeat()

Syntax:

for (var in set) { commands }

while (condition) { commands }

repeat { commands }

These statements are used to execute a set of commands recursively based on a logical condition or for a fixed no of times.

Examples:

```
x <- 0
for ( i in 1:5 ) { if (i==3) { next } ; x <- x + i }
y <- 1
j <- 1
while( y < 12 & j < 8 ) { y <- y*2 ; j <- j + 1}
z <- 3
repeat { z<- z^2; if ( z>100 ) { break }; print(z)}
```

Impact of Data outliers

Without Outlier	With Outlier
4, 4, 5, 5, 5, 5, 6, 6, 6, 7, 7	4, 4, 5, 5, 5, 5, 6, 6, 6, 7, 7,300
Mean = 5.45	Mean = 30.00
Median = 5.00	Median = 5.50
Mode = 5.00	Mode = 5.00
Standard Deviation = 1.04	Standard Deviation = 85.03

How can we treat for outliers ??

- Deleting observations
- Binning or transforming variables
- Imputation of outlier values

➢ **Use hist() function to identify outlier values**

What values can be considered as outliers ??

1. Any value, which is beyond the range of -1.5 x IQR to 1.5 x IQR
2. Use capping methods. Any value which out of range of 5th and 95th percentile can be considered as outlier
3. Data points, three or more standard deviation away from mean are considered outlier
4. Outlier detection is merely a special case of the examination of data for influential data points and it also depends on the business understanding