Session 1: Getting started (R basics)

Bishwa Ghimire

16.1.2019

Contents

1 Getting started **2**
1.1 R introduction and installation 2
1.2 R-studio installation and overview 2
1.3 Course legend 2
1.4 Ready to code 2
1.5 Operators 2
1.6 Variables 3
1.7 Data types 3
1.8 Special constants in R 9
1.9 File input/output 10
1.10 Control Structures 10
1.11 Saving objects and workspace 12
1.12 Some useful R functions 12
1.13 Answers 13

1 Getting started

1.1 R introduction and installation

R is free and open-source software for statistical computing and graphics. Go to https://www.r-project.org/ and download R binary depending on your operating system. In this course we use Rstudio as an IDE. Download it from https://www.rstudio.com. For instalaltion check presentation slides.

1.2 R-studio installation and overview

Check the presentation slides.

1.3 Course legend

1.4 Ready to code

Open R studio from application list and run `getwd()` in R console. It is important to set working directory in R. Once you set an working directory output files and plot can be saved in the location relative to the working directoty. You can set working directory using the function `setwd("/path/to/working/directory")` in unix based systems. In Windows backslash (\) is used in place of slash (/).

```
getwd()
```

1.5 Operators

Arithmetic operators take numeric values, evaluate them and result a single value. Following are the arithmetic operators used in R. In the following R code we evalute many numeric values to produce a single value. The modulus operator (%%) gives a remainder as output when a number is devided by another.

Arithmetic Operators	
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus	%%
Exponent	^

```
1 + 1 -7*2
2^8
4%%2
```

Logical operators typically take Boolean (eg TRUE/FALSE, 0/1) values and result single boolean value. Following are some of the logical operators used in R.

Logical Operators	
Not	!
AND	&
OR	|
Equality	==

Logical Operators	
NOT Equality	!=

```
!TRUE
TRUE & TRUE
TRUE & FALSE

TRUE | TRUE
FALSE | FALSE

TRUE == FALSE
1 == 2
1 != 1
```

In the above code the AND operator gives an output TRUE when both inputs are `TRUE`. Whenever one of the inputs is `FALSE`, the output of the operator is `FALSE`. The OR operator gives output `TRUE` when one of the inputs is `TRUE`. NOT operator inverts an input. If input is `TRUE`, output of the NOT operator is `FALSE` and vice versa. It takes only one input.

1.6 Variables

In R variable name can not start from number and can not contain spaces. Following are some convensions to name a variable. It is a good idea to select one of the styles in your coding. It is not recommended to use R keywords as variable name eg. `data`.

```
myvariable  <- 1
my_variable <- "2"
my.variable <- c(1, 2, 3)
myVariable  <- c(1:5)
MyVariable  <- 8
```

1.7 Data types

1.7.1 Basic data types

R has 5 basic data types.

- Character
- Numeric ($\mathbb{R}$)
- Integer ($\mathbb{Z}$)
- Complex ($\mathbb{C}$)
- Logical

```
my.var <- "A"
class(my.var)

my.var <- 1.5
class(my.var)

my.var <- integer(1)
class(my.var)

my.var <- 1+0i
class(my.var)
```

```
my.var <- TRUE
class(my.var)
```

The function `class()` is a very useful function to check a data type.

Following are additional data types in R.

- Vector
- Factors
- Data frame
- List
- Matrix
- Function

1.7.2 Vector

The function `c()` is used to create a vector. Vector is one dimensional data type.

```
my.vector <- c("a","b", "c")
my.vector <- c(1:5)
my.vector <- c(0.5, 1.6, 9.3)
my.vector <- c(TRUE, FALSE, TRUE, TRUE)
my.vector <- c(11+3i, 2+0i, 0+5i)

length(my.vector)
```

```
## [1] 3
```

```
## Can you figure out what happens when you run the following code?
my.vector <- c("Book", "Pencil", 5, 8, 9)
```

1.7.3 Factors

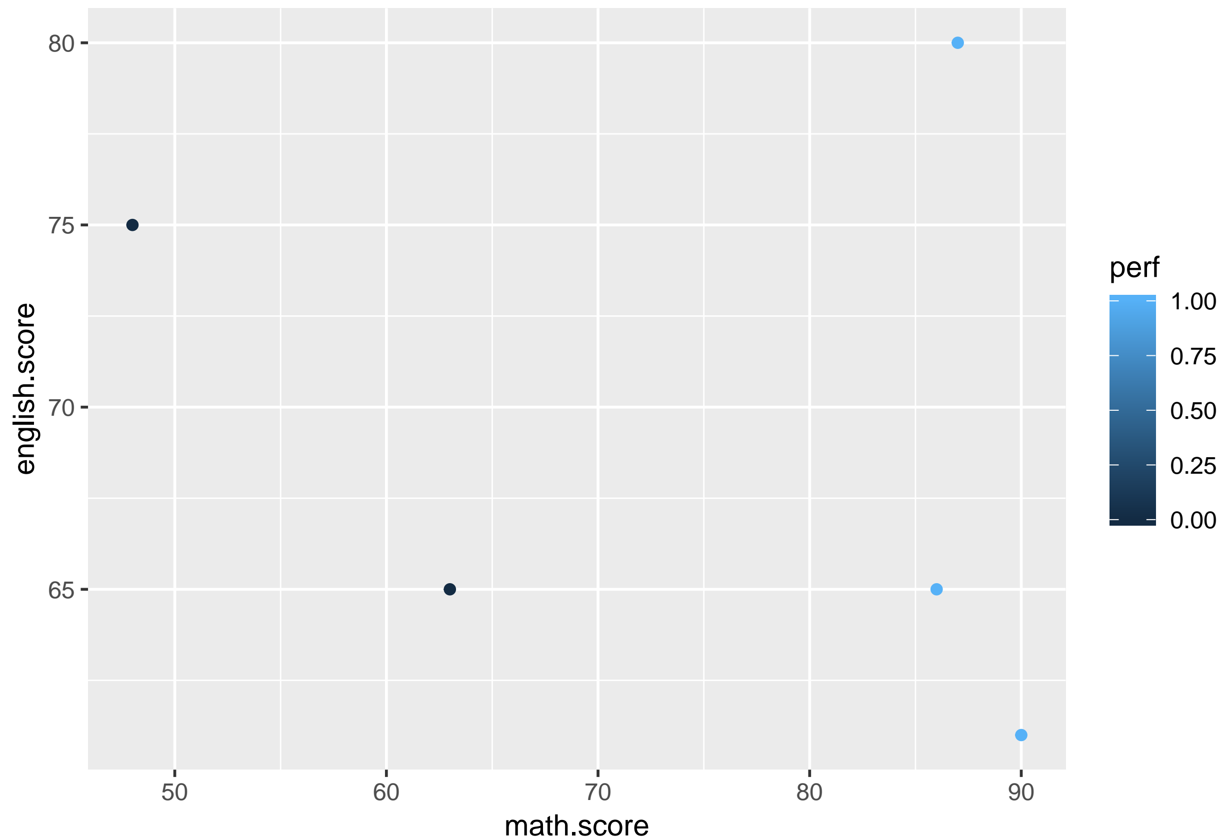
Factor is a categorical data in R. It can be order or unordered. It is very important to mark specific variable as categorical in statical modeling (eg. when using `lm()` function) and in making plots. If a `data.frame` column is character vector R can take it as a categorical variable, but if categories are encoded into integers, they should be converted into `factor`.

In the following code the main idea is not to understand every single line of code, but to understand importance of factor.

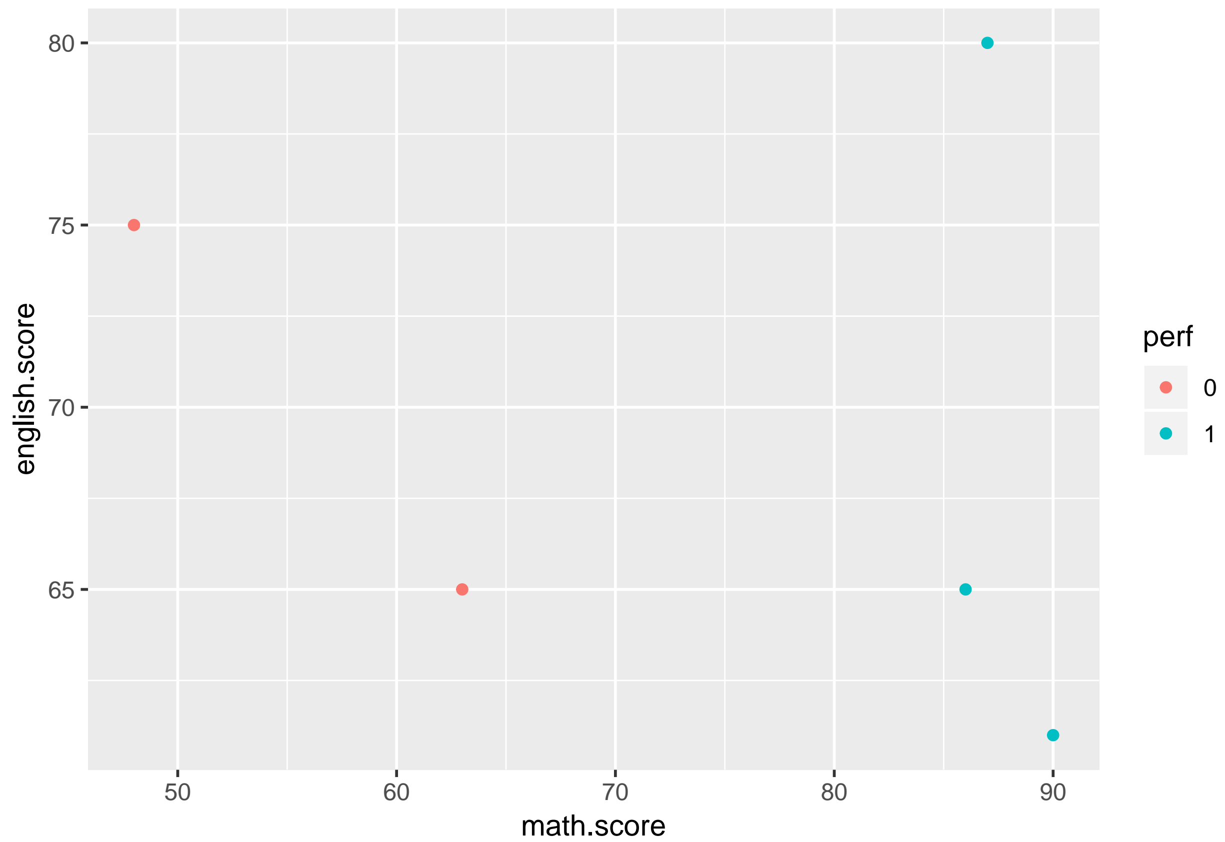
```
z <- c(1, 2 ,2 , 2, 8, 3, 6, 4, 5, 1, 1, 6, 5, 2, 7, 6, 5, 2, 7, 3, 8, 9, 2, 9, 7, 4, 10)
fac <- factor(z)



z.df <- data.frame(student = c("Mark", "Ana", "Peter", "John", "Jenny"),
                   math.score = c(63, 48, 86, 90, 87),
                   english.score = c(65, 75, 65, 61, 80),
                   physics.score = c(40, 55, 78, 71, 65))

## take average score per student and seggrigate into two groups based
## on threshold average score 60
z.df$avg <- rowMeans(z.df[,-1])
z.df$perf <- ifelse(z.df$avg<60,0,1)
```

In the following plot `perf` has a continuous color scale in legend which is not a good thing to have. Once we change `pref` to factor the legend turns into discrete class.

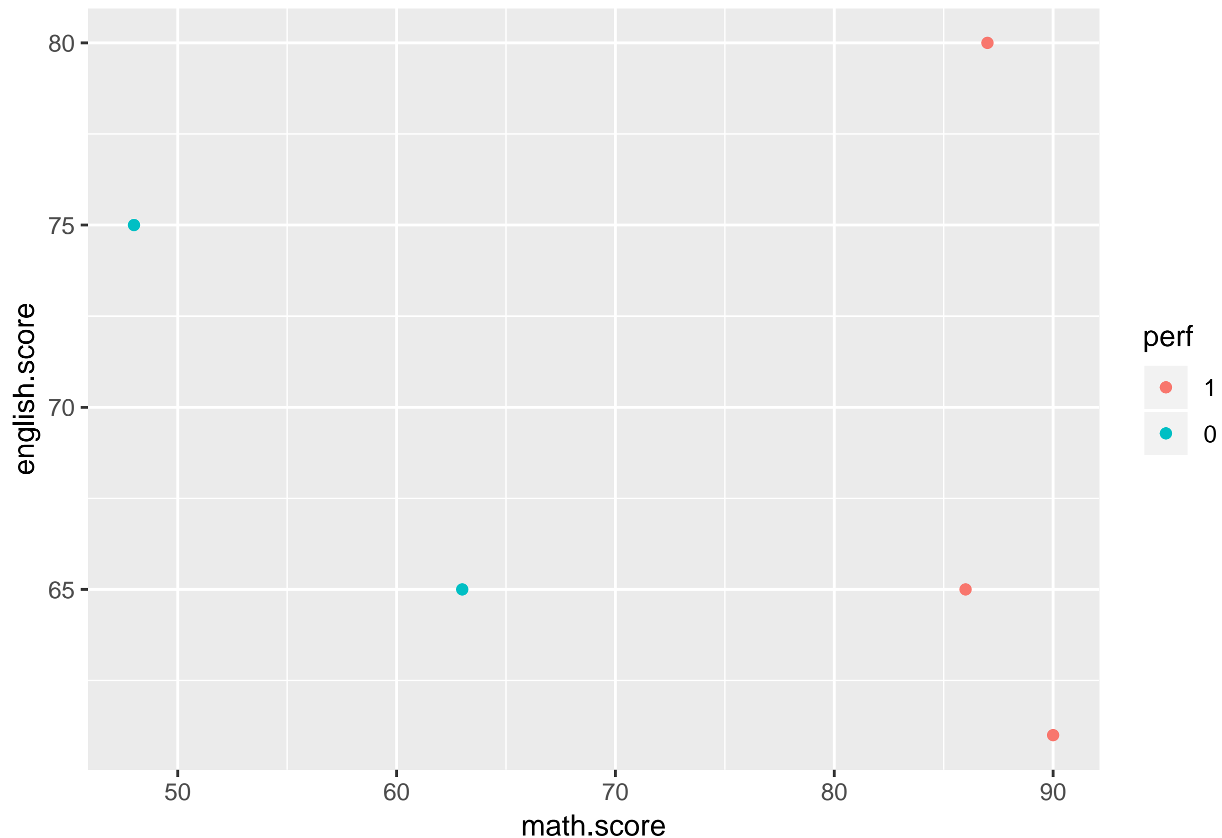
```r
library(ggplot2)
ggplot(z.df,aes(x=math.score, y=english.score, colour=perf))+geom_point()
```




```r
## lets convert perf column to factor
z.df$perf <- factor(z.df$perf)
ggplot(z.df,aes(x=math.score, y=english.score, colour=perf))+geom_point()
```




```
## lets order factor
z.df$perf <- factor(z.df$perf, levels = c(1,0))
ggplot(z.df,aes(x=math.score, y=english.score, colour=perf))+geom_point()
```



Assignment 1

What happens if you replace the line

`z.df$perf <- ifelse(z.df$avg<60,0,1)`

by

`z.df$perf <- ifelse(z.df$avg<60,"Below threshold","Above threshold")``

in the code above and run

`ggplot(z.df,aes(x=math.score, y=english.score, colour=perf))+geom_point()`

after that?

1.7.4 Data frame

Data frames are analogs to real world table. Data frame has two dimensional structure with rows and columns. It has unique row names and column names. Data frame columns can contain different data types.

```
my.df <- data.frame(student = c("Mark", "Ana", "Peter", "John", "Jenny"),
                    math.score = c(63, 49, 86, 90, 87),
                    english.score = c(71, 79, 65, 61, 80),
                    physics.score = c(40, 57, 78, 71, 65))
dim(my.df)
colnames(my.df)
rownames(my.df)

## show entire data frame
my.df
```

```
## show first row and first column
my.df[1,1]

## show first column
my.df[,1]

## show first row
my.df[1,]

## show first and second column
my.df[,1:2]

## show first three rows
my.df[1:3,]

## Create empty data frame
empty.df <- data.frame(student=character(),
                  math.score=numeric(),
                  english.score=numeric(),
                  physics.score=numeric())
```

1.7.5 List

```
## Creating a list
my.list <- list(name="Mark", age=22)
my.list

## Check names in list
names(my.list)
my.list$name

## change names
names(my.list) <- c("student.name", "age")

## empty list
empty.list <- vector("list", length = 4)

## Check data type
class(empty.list)
```

Assignment 2

Can you create lists inside a list?

1.7.6 Matrix

Matrix is two dimensional data type with rows and column but with same data type (numeric, character etc). You can use `as.matrix()` function to convert for example a numeric data frame.

```
my.matrix <- matrix(0, nrow=2, ncol=3)
my.matrix <- matrix(1:6, nrow=2, ncol=3)

rownames(my.matrix) <- c("R1", "R2")
```

```r
colnames(my.matrix) <- c("C1", "C2", "C3")

my.matrix

## retrive dimension names
dimnames(my.matrix)

## Create empty  matrix
empty.matrix <- matrix(nrow = 2, ncol = 3)

## Can you figure out what happened in the following line?
as.matrix( my.df )
```

1.7.7 Function

A function is a block of codes that does a specific task. A task could be taking mean of input numbers or something else. A function usually returns a value. For example `sum()` is a function. In the expression `sum(2,5)` it takes input values 2 and 3 which are called arguments and returns summed up value `5`. The function `return()` is used to return a value from a function. We can assign returned valued to a variable.

```r
## Using built-in R function
sum(2,3)
## Here we assign value returned by sum function to the variable named total.
total <- sum(2,3)
```

It is possible to create your own function.

```r
## Creating custom function
## Function definition
check.equal <- function(input1, input2)
{
  if(input1 == input2){
    return(TRUE)
  }else{
    return(FALSE)
  }
}

## Function call
check.equal(4,5)
check.equal(9,9)
```

Assignment 3

Create a function that accepts a number as an input and returns its square.

1.8 Special constants in R

- NA - Missing values.
- NaN - Value can not be determined.
- Nan is NA, but NA is not NaN.
- NA is logical.

- NaN is an integer.

```
## NA is logical
class(NA)

a <- c(5, 7, 8, NA, 2, 0, 12, NA)
b <- c(5, 7, 8, NA, 2, 0, 12, NaN)

is.na(a)
a[!is.na(a)]

## Nan is NA, but NA is not NaN.
is.na(b)
is.nan(b)

## NaN
0/0
```

1.9 File input/output

It is possible to read wide range of files (TXT, CSV, JSON, XML, SPSS etc) in R. In this course we focus on delimited text files. The functions `read.table()` or `read.csv()` can be used to read delimited text files.

```
## lets populate a data frame with random numbers and save to a file first.
my.df <- as.data.frame( matrix(rnorm(10*1000, 1, .5), ncol=10) )

## Write to tab delimited file.
## sep="\t" tells that the file is tab delimited
## quote=FALSE disables putting quotes on every value
write.table(my.df, file="output.tsv", sep="\t", quote=FALSE)

## Read tab delimited file.
my.df.2 <- read.table("output.tsv", sep="\t")
```

1.10 Control Structures

Control structures enables us to change control flow of your code.

1.10.1 If-else statements

If-else structure:

```
if(condition){
  Code block to run when the condition is TRUE
  } else{
  Code block to run when the condition is FALSE
  }
```

A condition should be either `TRUE` of `FALSE`. In the following code `x %% 5 == 0` gives `TRUE`. Thus if block of codes will be executed.

```
x <- 15

if(x %% 5 == 0){
  print("Exactly divisible")
}else{
  print("Not exactly divisible")
```

```
}

## If block and else if block can not run at the same time.
## Use only conditions are mutually exclusive
if(x < 16){
  print("Less than 16")
}else if(x + 15 > 20){
  print("Greatern than 20")
}else if(x + 15 > 18){
  print("Greater than 18")
}else{
  print("None of the above")
}
```

1.10.2 Loops

Loops are common way to perform iterations. In R `while`, `for` and `repeat` are used to create iterations.

```
## While loop
i<-0;
while (i<=10) {
  print(i)
  i <- i +1
}

## For loop
for(x in 0:10){
  print(x)
}

## Repeat loop
## repeat runs indifinitely if no there is no break()
m <- 0
repeat{
  m <- m+1
  print(m)
  if(m==10)
    break()
}
```

The function `break()` is used to terminate a loop and `next()` is used to skip a current iteration. In the following first loop the loop terminates as soon as the condition `x==5` is satisfied. In the second loop as soon as the condition `x==5` is satisfied the current iteration is skipped.

```
## Exit a loop
for(x in 0:10){
  if(x==5)
    break()
  print(x)
}

## skip an iteration
for(x in 0:10){
  if(x==5)
    next()
```

```
  print(x)
}
```

1.11 Saving objects and workspace

It is possible to save entire R workspace, command history and objects. Upon quit R prompt asks whether or not to save workspace. A workspace can be saved manually using `save.image()`. If no filename is provided, `.RData` file is created in the current working directory.

```
## Saving R workspace
save.image()
save.image(file = "workspace-1.RData")

## Save R history
savehistory()
savehistory(file = "workspace-1.Rhistory")
```

R objects can be saved/loaded using two functions `save()`/`load()` and `saveRDS()`/`readRDS()`. The function `save()` loads the saved object to the current workspace. Therefore, the object can be accessed using the same variable.

```
my.var <- rnorm(50, mean=5, sd=1)
save(my.var, file="my.var.RData")
rm(my.var)
load(file="my.var.RData")
my.var
```

However, same thing does not happen when you use `readRDS()` to load an object saved using `saveRDS()`. The function does not load the object but reads the object and returns its content. Therefore, another variable/object is needed to hold an output of `readRDS()`.

```
my.var2 <- rnorm(50, mean=10, sd=2)
saveRDS(my.var2, file="my.var2.RDS")
var <- readRDS("my.var2.RDS")
rm(my.var2)
var
```

1.12 Some useful R functions

Following are some fo the commonly used functions.

```
## get current working directory
getwd()

## List directories in current directory
dir()

## check functions of a package
library(MASS)
ls("package:MASS")
help(package=MASS)

## help
?area
help(area)
```

```
## concatinate string
a <- "Hello"
b <- "World!"

paste(a, b, sep=" ")
```

1.13 Answers

Assignment 1: What happens if you replace the line

`z.df$perf <- ifelse(z.df$avg<60,0,1)`

by

`z.df$perf <- ifelse(z.df$avg<60,"Below threshold","Above threshold")`

in the code above and run

`ggplot(z.df,aes(x=math.score, y=english.score, colour=perf))+geom_point()`

after that?

Answer: This modified line adds character vectors to the coulmn `z.df$avg`, which does not need factor conversion. Thus, produces correct legend in the plot.

Assignment 2: Can you create lists inside a list? How to access such list?

Answer:

```
my.lst <- list(record1=list(first.name="John", last.name="Harrison", age=25),
     record2=list(first.name="Peter", last.name="Smith"), age=20)

my.lst$record1
my.lst[1]

my.lst$record1$first.name
my.lst[[1]][1]
```

Assignment 3: Create a function that accepts a number as an input and returns its square.

Answer:

```
my.fun <- function(input){
  return(input^2)
}
```