bioinformatics.ca

Canadian Bioinformatics Workshops

www.bioinformatics.ca

R Review

bioinformatics.ca

florence — R — 77×44

```
Last login: Fri May 16 17:29:10 on ttys008
Florences-MacBook-Pro:~ florence$ R

R version 3.1.0 (2014-04-10) -- "Spring Dance"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (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.

> a<- c(1:10)
> a
 [1]  1  2  3  4  5  6  7  8  9 10
> log2(2)
[1] 1
> log2(a)
 [1] 0.000000 1.000000 1.584963 2.000000 2.321928 2.584963 2.807355 3.000000
 [9] 3.169925 3.321928
> 
```

Modified from Richard De Borja, Cindy Yao and Florence Cavalli

Objectives

- To review the basic commands in R
- To review the matrix, data frame and list objects
- To learn more about how to visualize our data

➢ To become more familiar with R!

Survey

- What is your experience in R?
  - Never used at all
  - Used for the first time to prepare the workshop
  - Used in my work but very basic
  - Using it rather regularly in my work
- Do you have experience in any other language?
  - Java, C, C++, …
  - Perl, Python, Ruby, …

What is R?

R is a programming language and software environment for statistical computing and graphics

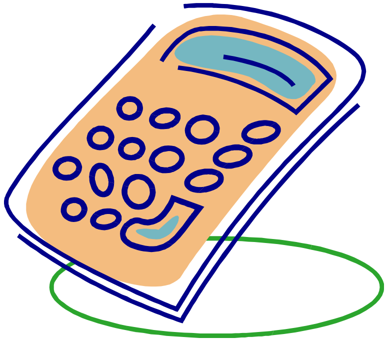
R allows for:

- Data handling and manipulation
- Statistical tests
- Graphics
- Specialized data analysis (i.e. microarray data, Seq data)

An Overgrown Calculator

- Everything you can do on a calculator you can do in R plus more!

```
> 2+5
[1] 7
> 30/10
[1] 3
> 20-5
[1] 15
> log10(20)
[1] 1.30103
> pi
[1] 3.141593
> exp(-2)
[1] 0.1353353
> sin(0.4)
[1] 0.3894183
```



Assignments

- The arrow **<-** is the assignment operator

```
> weight.a <- 10
> weight.a
[1] 10
> weight.b <- 30
> weight.b
[1] 30
> total.weight <- weight.a + weight.b
> total.weight
[1] 40
```

Tips:

- Avoid single letter names
- Separate words with a period or uppercase letters (i.e. total.weight or totalWeight)

Code Documentation

```
# Hi there! How are you?

# This is a comment!
```

- What is it for?
  - Explain what you are going to do with the code
  - Write messages for yourself
- Proper documentation is important!

```
# calculate the sum of 3 numbers
sum(c(2, 6, 8))
```

Working Directory

- In which directory are you working?

  getwd()

```
> getwd()
[1]
"/Users/florence/Canadian_Bioinfo_workshop/BiCG_workshop_2014/R_rev
iew_2014"
```

- How can you change the working directory?

  setwd()

```
> setwd("C:/myPATH")
> setwd("~/myPATH") # on Mac
> setwd("/Users/florence/myPATH") # on Mac
```

- File list in working directory, and object list in R:

  list.files(); ls()

Finding help: within R

- Call help on a function
  - help(sum)
  - ?sum
- Read the documentation
  - See next slide
- Quit the help
  - q
- A more general search
  - help.search(“plot”)
  - ??plot

Name
of the function

Description of
the function

How to use it

Information
regarding the
function

```
sum                        package:base                        R Documentation

Sum of Vector Elements

Description:

     'sum' returns the sum of all the values present in its arguments.

Usage:

     sum(..., na.rm = FALSE)

Arguments:

     ...: numeric or complex or logical vectors.

   na.rm: logical.  Should missing values (including 'NaN') be removed?

Details:

     This is a generic function: methods can be defined for it directly
     or via the 'Summary' group generic.  For this to work properly,
     the arguments '...' should be unnamed, and dispatch is on the
     first argument.
```

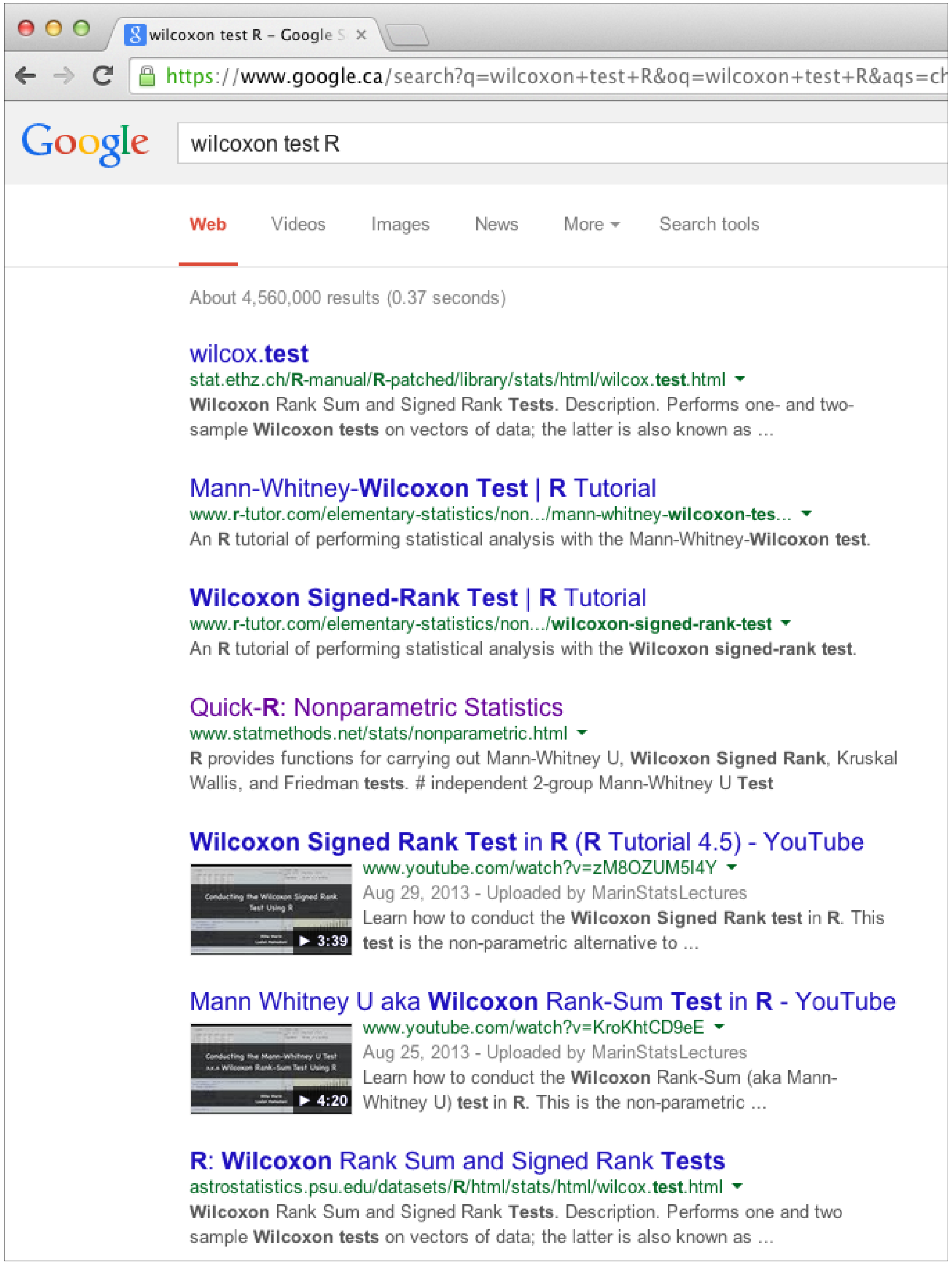
Finding help: useR

- R website http://www.r-project.org
  - Documentation
  - Mailing-list R-help
- Useful links
  - http://www.rseek.org
  - http://www.r-bloggers.com
  - http://biostar.stackexchange.com
- Colleagues
- Local groups
  - GTA useR group

Finding help: Google & co

Many blogs, tutorial, comments online, …

=> General web search engines are useful for that



Vectors

- Vectors are a collection of elements of the same data type
  - Numeric

```
> numeric.vector
[1] 1 2 3 4 5 6 2 1
```

  - Character

```
> character.vector
[1] "Fred"   "Barney" "Wilma"  "Betty"
```

  - Logical

```
> logical.vector
[1]  TRUE  TRUE FALSE  TRUE
```

Creating Vectors

- The *c()* function can be used to combine arguments and create vectors; str() function can be used to check the structure of the object

```
> numeric.vector <- c(1,2,3,4,5,6,2,1)
> numeric.vector
[1] 1 2 3 4 5 6 2 1

> character.vector <- c("Fred", "Barney", "Wilma", "Betty")
> character.vector
[1] "Fred"   "Barney" "Wilma"  "Betty"

> logical.vector <- c(TRUE, TRUE, FALSE, TRUE)
> logical.vector
[1]  TRUE  TRUE FALSE  TRUE

# check the structure of the object:
> str(logical.vector)
 logi [1:4] TRUE TRUE FALSE TRUE
```

Vector Indexing

- Use the position in the vector to select value of interest with the operator [ ]

```
> character.vector
[1] "Fred"   "Barney" "Wilma"  "Betty"

> character.vector[2]
[1] "Barney"

> character.vector[2:3]
[1] "Barney" "Wilma"

> character.vector[c(2,4)]
[1] "Barney" "Betty"
```

Factors

- Factors store categorical data (i.e. gender)

```
> gender <- c(1,2,1,1,1,2)
> gender
[1] 1 2 1 1 1 2

> gender.factor <- as.factor(gender)
> gender.factor
[1] 1 2 1 1 1 2
Levels: 1 2

> levels(gender.factor) <- c("male", "female")
> gender.factor
[1] male   female male   male   male   female
Levels: male female
```

Factor Indexing

- Indexing a factor is the same as indexing a vector

```
> gender.factor
[1] male   female male   male   male   female
Levels: male female

> gender.factor[2]
[1] female
Levels: male female

> gender.factor[2:4]
[1] female male   male
Levels: male female

> gender.factor[c(1,4)]
[1] male male
Levels: male female
```

Matrix

- Matrices are tables of numbers

```
> ?matrix
##(…)
Usage:
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
            dimnames = NULL)

> matrix.example <- matrix(1:12, nrow = 3, ncol=4, byrow = FALSE)
> matrix.example
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

> matrix.example <- matrix(1:12, nrow = 3, ncol=4, byrow = TRUE)
> matrix.example
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

Creating Matrices

- The *rbind()* and *cbind()* functions can be used to combine vectors and create matrices. This is equivalent to the *c()* function for vectors

```
> dataset.a <- c(1,22,3,4,5)
> dataset.b <- c(10,11,13,14,15)
> dataset.a
[1]  1 22  3  4  5
> dataset.b
[1] 10 11 13 14 15

> rbind.together <- rbind(dataset.a, dataset.b)
> rbind.together
          [,1] [,2] [,3] [,4] [,5]
dataset.a    1   22    3    4    5
dataset.b   10   11   13   14   15

> cbind.together <- cbind(dataset.a, dataset.b)
```

Creating Matrices

```
> dataset.a <- c(1,22,3,4,5)
> dataset.b <- c(10,11,13,14,15)
> dataset.a
[1]  1 22  3  4  5
> dataset.b
[1] 10 11 13 14 15

> rbind.together <- rbind(dataset.a, dataset.b)
> rbind.together
          [,1] [,2] [,3] [,4] [,5]
dataset.a    1   22    3    4    5
dataset.b   10   11   13   14   15

> cbind.together <- cbind(dataset.a, dataset.b)
> cbind.together
     dataset.a dataset.b
[1,]         1        10
[2,]        22        11
[3,]         3        13
[4,]         4        14
[5,]         5        15
```

Matrix Indexing

- Use the row and column positions to select value of interest with the operator [ ]

  i.e matrixObject[row_id,column_id]

```
> matrix.example
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12

> matrix.example[2,4]
[1] 8
> matrix.example[2,]
[1] 5 6 7 8
> matrix.example[,4]
[1]  4  8 12
```

Matrix Indexing

- Re-name rows and columns and select values of interest

```
> colnames(matrix.example) <- c("Sample1","Sample2","Sample3","Sample4")
> rownames(matrix.example) <- paste("gene",1:3,sep="_")

> matrix.example
       Sample1 Sample2 Sample3 Sample4
gene_1       1       2       3       4
gene_2       5       6       7       8
gene_3       9      10      11      12

> matrix.example[,"Sample2"]
gene_1 gene_2 gene_3
     2      6     10
> matrix.example[1,"Sample2"]
[1] 2
> matrix.example["gene_1","Sample2"]
[1] 2
```

Data frames

- Data frames are similar to matrices but each column can be a different data type

```
> people.summary <- data.frame(
+ age = c(30,29,25,25),
+ names = c("Fred", "Barney", "Wilma", "Betty"),
+ gender = c("m", "m", "f", "f")
+ )

> people.summary
  age  names gender
1  30   Fred      m
2  29 Barney      m
3  25  Wilma      f
4  25  Betty      f
```

Data frame Indexing

- Indexing a data frame can be done the same way you index a matrix
  You can also use the $ to obtain a column

```
> people.summary
  age  names gender
1  30   Fred      m
2  29 Barney      m
3  25  Wilma      f
4  25  Betty      f
```

```
> people.summary[2,1]
[1] 29

> people.summary[2,]
  age  names gender
2  29 Barney      m

> people.summary[,1]
[1] 30 29 25 25

> people.summary$age
[1] 30 29 25 25
```

Lists

- Lists are combinations of data which can vary in data type and length

```
> together.list <- list(
+ vector.example = dataset.a,
+ matrix.example = matrix.example,
+ data.frame.example = people.summary
+ )
> together.list
$vector.example
[1]  1 22  3  4  5

$matrix.example
       Sample1 Sample2 Sample3 Sample4
gene_1       1       2       3       4
gene_2       5       6       7       8
gene_3       9      10      11      12

$data.frame.example (to continue)
```

Lists

```
> together.list <- list(
+ vector.example = dataset.a,
+ matrix.example = matrix.example,
+ data.frame.example = people.summary
+ )
> together.list
$vector.example
[1]  1 22  3  4  5

$matrix.example
       Sample1 Sample2 Sample3 Sample4
gene_1       1       2       3       4
gene_2       5       6       7       8
gene_3       9      10      11      12

$data.frame.example
  age  names gender
1  30   Fred      m
2  29 Barney      m
3  25  Wilma      f
4  25  Betty      f
```

List Indexing

- You can index a list by using the $, [] , or [[]]

```
> together.list$matrix.example
       Sample1 Sample2 Sample3 Sample4
gene_1       1       2       3       4
gene_2       5       6       7       8
gene_3       9      10      11      12

> together.list$matrix.example[,3]
gene_1 gene_2 gene_3
     3      7     11
```

```
> together.list["matrix.example"]
$matrix.example
       Sample1 Sample2 Sample3 Sample4
gene_1       1       2       3       4
gene_2       5       6       7       8
gene_3       9      10      11      12

> together.list[["matrix.example"]]
       Sample1 Sample2 Sample3 Sample4
gene_1       1       2       3       4
gene_2       5       6       7       8
gene_3       9      10      11      12

> together.list[["matrix.example"]][,2]
gene_1 gene_2 gene_3
     2      6     10
```

- [ ] allows you to select multiple elements
- $ and [[ ]] allows you to select a single element

Functions

- Functions are a set of commands that work together to perform a given task
- Arguments are parameters you provide to the function for processing

Most functions have reasonable default values

```
> sum(c(1,2,3))
[1] 6
> log2(10)
[1] 3.321928
> sin(0.24)
[1] 0.2377026
> mean(c(1,2,3,4,5))
[1] 3
```

Some useful functions such as:

- Length of a vector: *length()*
- Number of rows or columns and dimension of a matrix/data frame: *nrow(), ncol(), dim()*

```
> character.vector
[1] "Fred"   "Barney" "Wilma"  "Betty"
> length(character.vector)
[1] 4
> matrix.example
       Sample1 Sample2 Sample3 Sample4
gene_1       1       2       3       4
gene_2       5       6       7       8
gene_3       9      10      11      12
> nrow(matrix.example)
[1] 3
> ncol(matrix.example)
[1] 4
> dim(matrix.example)
[1] 3 4
```

Some useful functions such as:

- Read a table for text file: *read.table()*
- Write a matrix/data frame in a text file: *write.table()*

```
> ?read.table
read.table("myDataFile.txt", header=TRUE, sep="\t",
stringsAsFactors=FALSE)

> ?write.table

> write.table(people.summary, file="File_name_people_summary.txt",
quote=FALSE, sep = "\t",row.names = FALSE, col.names = TRUE)
```

logical: should character vectors be converted to factors?

Let’s try! (I)

- Use the assignment operator to store three values of your choice
- Calculate and store the sum of the values from above

Solutions

- Use the assignment operator to store three values on your choice

```
> value.a <- 10
> value.b <- 3
> value.c <- 12
```

- Calculate and store the sum of the values from above

```
> sum.values <- value.a + value.b + value.c
```

Let’s try! (II)

We will use the *cars* dataset (included by default in R)

- What data type is the *cars* dataset? and its dimensions?
- Access to the speed values
- Access and store only the *cars* data with speeds greater than 15. How many cars does this affect?
- Reformat the *cars* data into a list
- Access only the *cars* data with speeds greater than 15 from the list you just created. How many cars does this affect? Did you get the same results as above?
- What does *stringsAsFactors* in the data.frame function do?

Solutions

- What data type is the *cars* dataset? and its dimensions?

```
> class(cars)
[1] "data.frame"
> dim(cars)
[1] 50  2
## or
> str(cars)
'data.frame':   50 obs. of  2 variables:
 $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
 $ dist : num  2 10 4 22 16 10 18 26 34 17 …
```

```
## To look at the object
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
```

- Access to the speed values

```
> cars$speed
 [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 14 14 14 15 15
[26] 15 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 20 22 23 24 24 24 24 25
```

Solutions

- Access and store only the *cars* data with speeds greater than 15
  How many cars does this affect?

```
> cars.greater.speed <- cars[cars$speed > 15,]
> nrow(cars.greater.speed)
[1] 24
```

- Reformat the *cars* data into a list

```
> cars.as.list <- list(SPEED = cars$speed, DISTANCE = cars$dist)
> cars.as.list
$SPEED
 [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 14 14 14 15 15
[26] 15 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 20 22 23 24 24 24 24 25

$DISTANCE
 [1]   2  10   4  22  16  10  18  26  34  17  28  14  20  24  28  26  34  34  46
[20]  26  36  60  80  20  26  54  32  40  32  40  50  42  56  76  84  36  46  68
[39]  32  48  52  56  64  66  54  70  92  93 120  85

> names(cars.as.list)
[1] "SPEED"    "DISTANCE"
```

Solutions

- Access only the *cars* data with speeds greater than 15 from the list you just created
  How many cars does this affect?
  Did you get the same results as above?

```
> cars.as.list.greater.speed <- cars.as.list$SPEED[cars.as.list$SPEED > 15]

> cars.as.list.greater.speed
 [1] 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 20 22 23 24 24 24 24 25

> length(cars.as.list.greater.speed)
[1] 24
```

Solutions

- What does *stringsAsFactors* in the data.frame function do?

```
> ?data.frame
##(…)
logical: should character vectors be converted to factors?
The ‘factory-fresh’ default is TRUE, but this can be
changed by setting options(stringsAsFactors = FALSE).
```

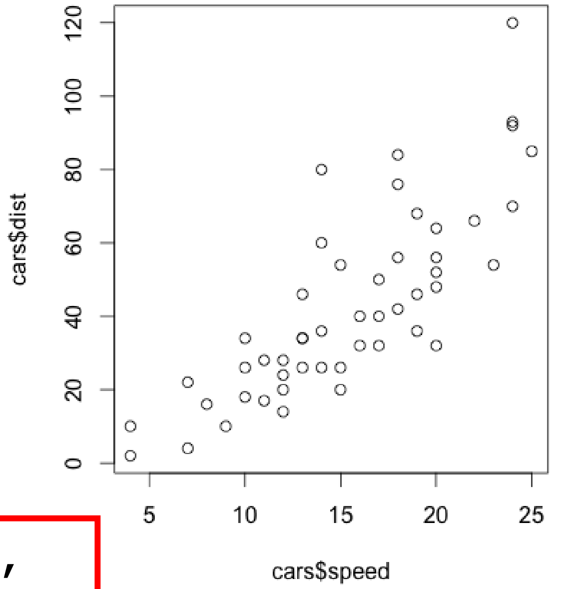
Exploratory analysis and plots

- A few tips:
  - Do not show too much information on a plot
  - Think about what message you want to give thanks to the plot
  - Avoid 3D graphics
  - Stay away from Excel (not a statistics package)

Let’s have a look: simple plot

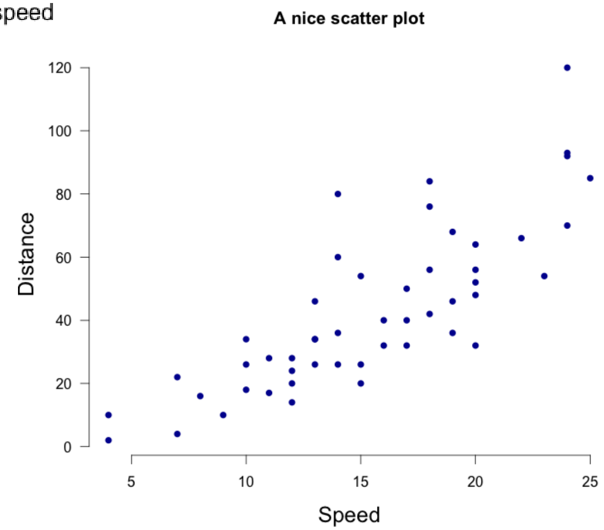
- Function *plot()*

```
> plot(x=cars$speed,
    y=cars$dist)
```



- Make it nicer

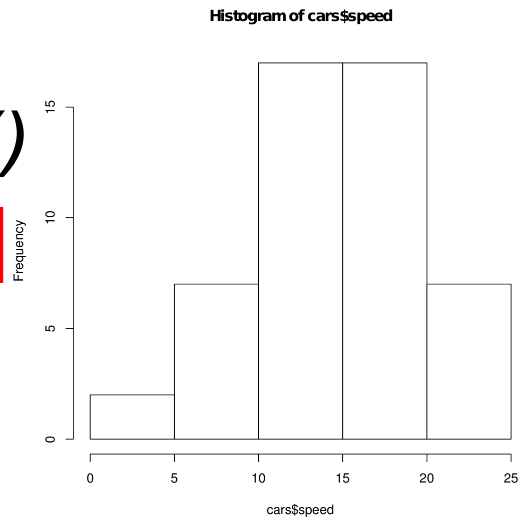
```
> plot(x=cars$speed, y=cars$dist,
    xlab = "Speed",
    ylab = "Distance",
    cex.lab = 1.5,
    main = "A nice scatter plot",
    pch = 16,
    bty = "n",
    col = "dark blue",
    las = 1)
```



Let’s have a look: histogram

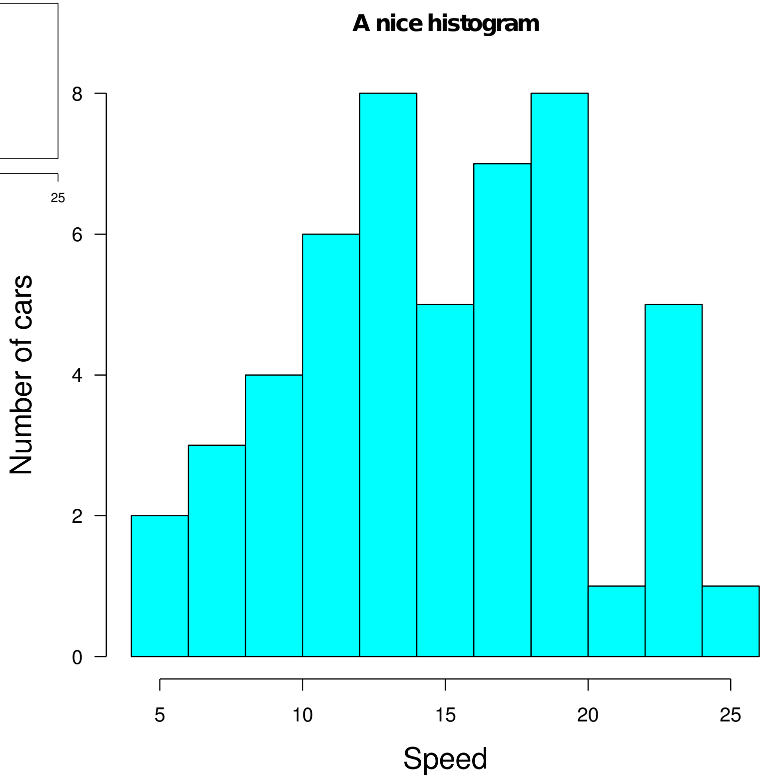
- Function *hist()*

```
> hist(cars$speed)
```



- Make it nicer

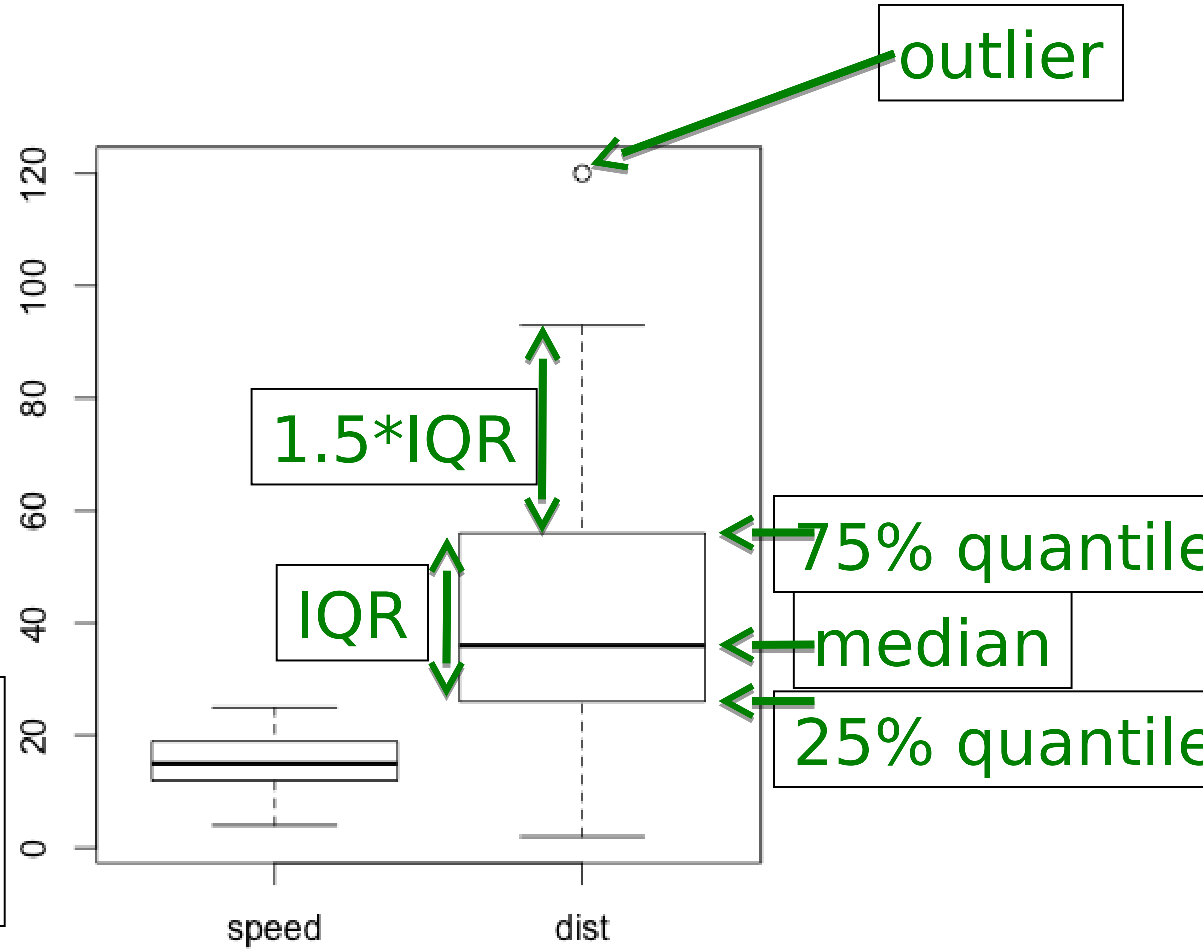
```
> hist(cars$speed,
    xlab = "Speed",
    ylab = "Number of cars",
    cex.lab = 1.5,
    main = "A nice histogram",
    col = "cyan",
    breaks = 10,
    las = 1)
```



Let's have a look: boxplot

- Function *boxplo*

```
> boxplot(cars)
```



IQR:
Inter Quantile Range
25%-75% quantile

Other useful functions related to figures

- Function *par()*
  - Allow to set many graphical parameters such as *mfrow, bg, col*,…
  - See *?par*
- Function *pdf()* then *dev.off()*
  - To save your plot as a .pdf figure

```
?pdf
##(…)
Description:

     ‘pdf’ starts the graphics device driver for producing PDF
     graphics.
```

Missing values

- R deals with missing values in a object using the *NA* value
- We can detect NA values with the *is.na()* function

```
> val <- c(1,3,5,NA,3,6)
> val
[1]  1  3  5 NA  3  6

> is.na(val)
[1] FALSE FALSE FALSE  TRUE FALSE FALSE

> which(is.na(val))
[1] 4
```

Let's try! (IV)

```
val <- c(1,3,5,NA,3,6)
```

- Compute the sum of *val* removing missing values
- I want the average of *val*, and I do mean(*val*)

 I am not happy with the result. What can I do?

Solutions

- Compute the sum of x removing missing values

```
> val <- c(1,3,5,NA,3,6)
> sum(val, na.rm = TRUE)
[1] 18
```

- I want the average of x, and I do mean(x)

  I am not happy with the result

  What can I do?

```
> mean(val)
[1] NA
> mean(val, na.rm = TRUE)
[1] 3.6
```

How to save and reload your data?

- Use the functions save() or save.image() and load()

```
> ?save
##(…)
Description:
     ‘save’ writes an external representation of R objects to the
     specified file.  The objects can be read back from the file at a
     later date by using the function ‘load’ (or ‘data’ in some cases).

     ‘save.image()’ is just a short-cut for ‘save my current
     workspace’, i.e., ‘save(list = ls(all = TRUE), file = ".RData")’.
     It is also what happens with ‘q("yes")’.

> save(cars.as.list, file="my_cars_as_list.RData")
> load(file="my_cars_as_list.RData") #if the file is present in the working directory, if not, indicate the path of the .RData file

> save(cars.as.list, numeric.vector, rbind.together,
file="my_Objects_Rreview_May2014.RData")
> load(file="my_Objects_Rreview_May2014.RData")

## To save all the objects in the R session
> save.image(file="Rreview_2014.RData")
## after closing you R session for example, load the data with:
> load(file="Rreview_2014.RData")
```

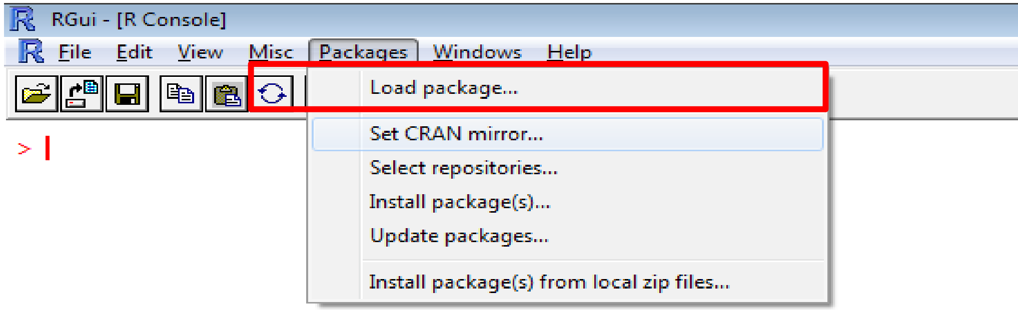
A few words about packages

- http://cran.r-project.org/
- "Currently, the CRAN package repository features 5563 available packages”
- To install a package: *install.packages()*

```
## install.packages("PackageName")
> install.packages("heatmap.plus")
```

- How to set your CRAN?

```
> chooseCRANmirror()
```



A few words on Bioconductor

- Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data.
- Site: http://bioconductor.org/
- Contains method, dataset and annotation packages
- May 2014: 824 software packages !
- To install a Bioconductor package:

```
> source("http://bioconductor.org/biocLite.R")
## biocLite("PackageName")
> biocLite("DESeq2")
> library("DESeq2")
```

Any questions?

Try and test by yourself!

Extra slides

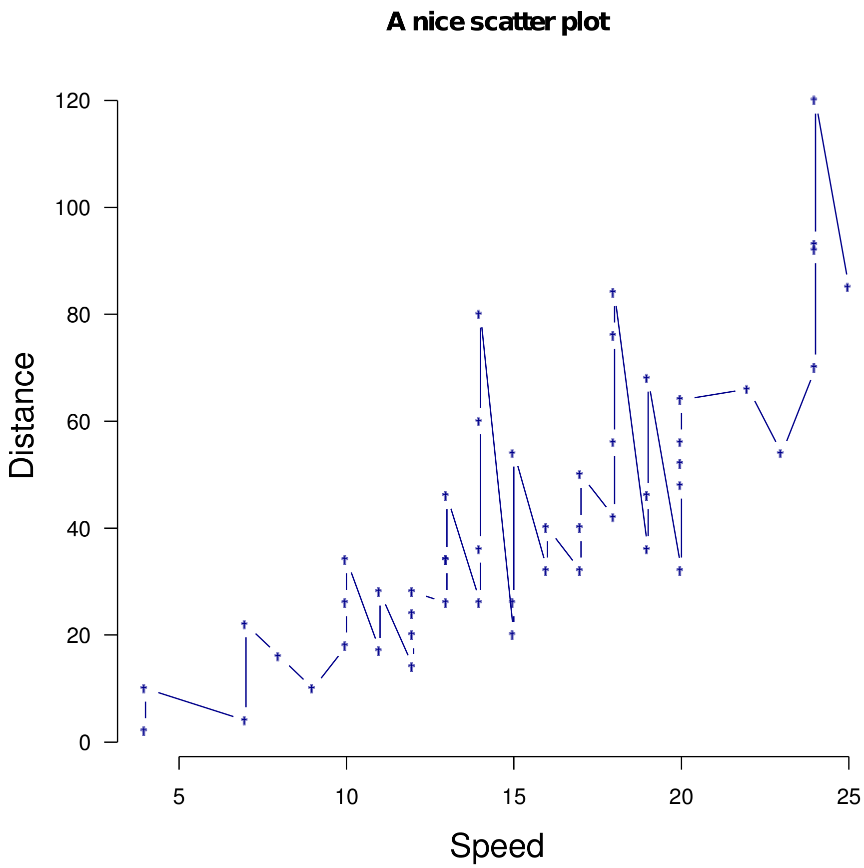
Let's try plotting! (III)

- Do a scatter plot with connected dots
- Make your customized version of the boxplot
- How can you change the 1.5*IQR parameter?
- Print the scatter plot and the boxplot on top of each other and save the figure in a pdf file

Solutions

- Do a scatter plot with connected dots

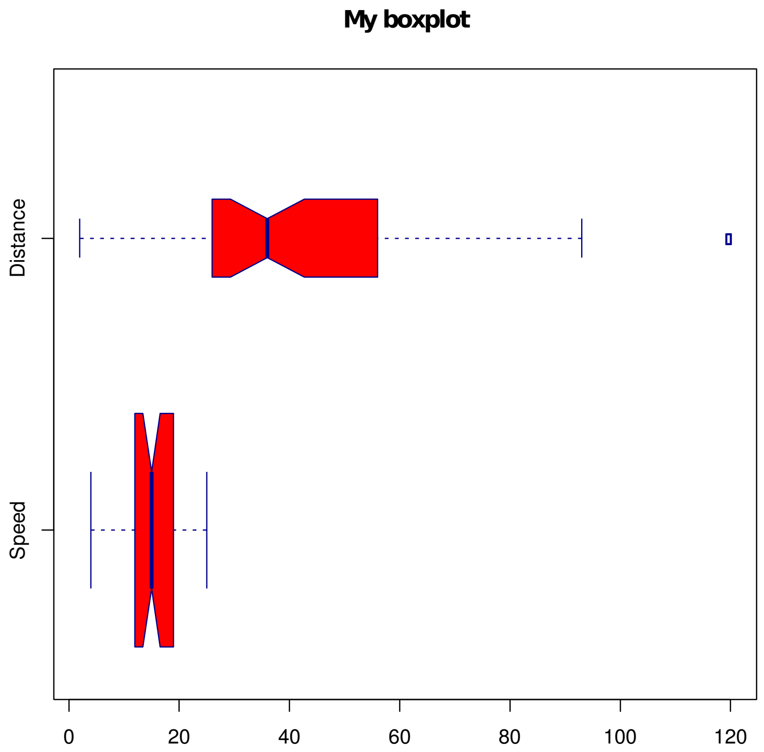
```
> plot(cars$speed, cars$dist,
    xlab = "Speed",
    ylab = "Distance",
    cex.lab = 1.5,
    main = "A nice scatter plot",
    pch = 16,
    bty = "n",
    col = "dark blue",
    type = "b",
    las = 1)
```



Solutions

- Make your customized version of the boxplot

```
> boxplot(cars,
    width = c(3,1),
    col = "red",
    border = "dark blue",
    names = c("Speed", "Distance"),
    main = "My boxplot",
    notch = TRUE,
    horizontal = TRUE)
```

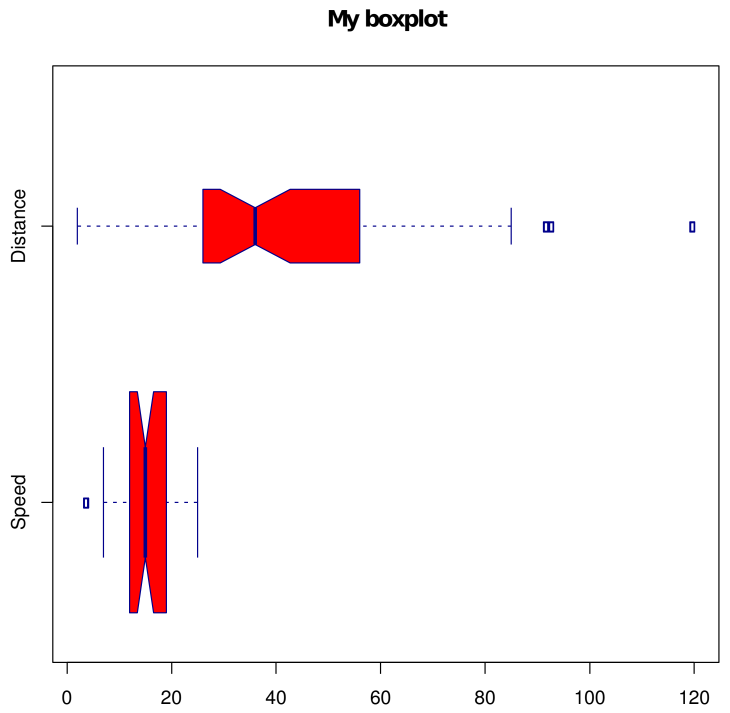


Solutions

- How can you change the 1.5*IQR parameter?

```
> boxplot(cars,
    width = c(3,1),
    col = "red",
    border = "dark blue",
    names = c("Speed", "Distance"),
    main = "My boxplot",
    range = 1,
    notch = TRUE,
    horizontal = TRUE)
```

range, default =1.5



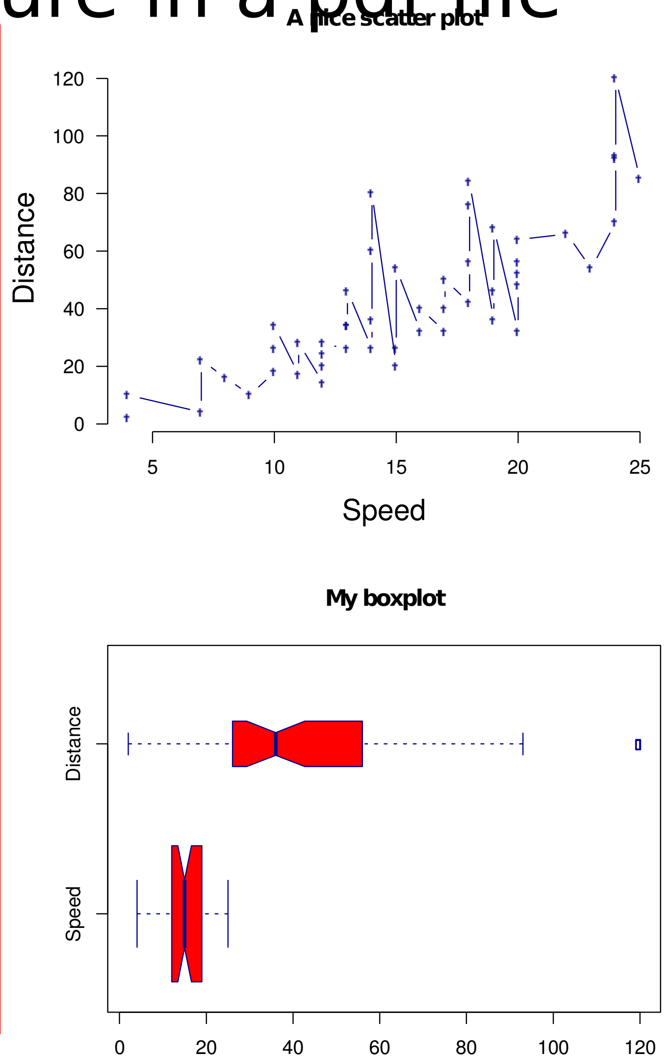
Solutions

- Print the scatter plot and the boxplot on top of each other and save the figure in a pdf file

```
> pdf("myfigure.pdf", height=18, width=8)
> par(mfrow=c(2,1))

> plot(cars$speed, cars$dist,
      xlab = "Speed", ylab = "Distance",
      cex.lab = 1.5,
      main = "A nice scatter plot",
      pch = 16,
      bty = "n",
      col = "dark blue",
      type = "b",
      las = 1)

> boxplot(cars,
         width = c(3,1),
         col = "red",
     border = "dark blue",
     names = c("Speed", "Distance"),
     main = "My boxplot",
     notch = TRUE,
     horizontal = TRUE)
> dev.off()
```



Statement control: If

- If/else

```
x <- 2
if (x>0) {
 cat("Positive value:",x,"\n")
} else if (x<0) {
 cat("Negative value:",x,"\n")
}
```

```
Positive value: 2
```

"\n" is to go to the next line

- If/else if/else

```
x <- -3
if (x>0) {
 cat("Positive value:",x,"\n")
} else if (x==0) {
 cat(“Zero:”,x,"\n”)
} else {
 cat("Negative value:",x,"\n")
}
```

```
Negative value: -3
```

Loop, loop, loop: For

- Indexes

```
for (i in 1:5) {
 cat(i)
}
```

```
12345>
```

- Vectors

```
values <- c(2,1,3,6,5)
for (value in values) {
## cat(value)
 print(value)
}
```

```
[1] 2
[1] 1
[1] 3
[1] 6
[1] 5
```

Loop, loop, loop: While

- Easy syntax…

Make sure something changes in each loop

```
x <- 4
while (x>0) {
 cat(“positive value:”,x,”\n”)
 x <- x-1
}
```

```
positive value: 4
positive value: 3
positive value: 2
positive value: 1
```

- And easy mistakes…

Oops.. we have a problem here… Infinite loop

```
x <- 4
while (x>0) {
 cat(“positive value:”,x,”\n”)
 x <- x+1
}
```

Make sure that the end criteria will happen

Let’s try! (V)

- Print all numbers from 1 to 10
- Print all even numbers from 1 to 10
- Print the speed of the first 8 cars using *while*
- Print the first 8 cars that have a speed more than 9
- How many cars have a speed greater than 10?
  What is their mean distance?

One way to do it with what we learned

- Print all numbers from 1 to 10

```
# create “sample” to browse
sample <- 1:10
for (n in sample) {
    # print each number in “sample”
    print (n)
    }
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

- Print all even numbers from 1 to 10

```
# create the sample to browse
sample <- 1:10
for (n in sample) {
    # test the rest of the division by 2 (see if even)
    if (n %% 2 == 0) {
        print (n)
        }
    # no need for a else here (and it is not required)
    }
```

```
[1] 2
[1] 4
[1] 6
[1] 8
[1] 10
```

One way to do it with what we learned

- Print the speed of the first 8 cars using *while*

```
# we initialize the index to track how many cars were printed
# 1 to start at the first car
index <- 1
# we continue until the index is 8
while (index <= 8) {
    # to access the speed of the car
    speed <- cars[index, "speed”]
    # we print with \n to go to the next line
    cat("Car #", index, "speed:", speed, "\n”)
    # every iteration we go to the next car
    index <- index + 1
    }
```

```
Car # 1 speed: 4
Car # 2 speed: 4
Car # 3 speed: 7
Car # 4 speed: 7
Car # 5 speed: 8
Car # 6 speed: 9
Car # 7 speed: 10
Car # 8 speed: 10
```

One way to do it with what we learned

- Print the first 8 cars that have a speed more than 9

```
# we initialize the variables so that they can be used in the loop
# 0 on the counter to add up
counter <- 0
# each time we find an appropriate car, 1 on the index to start at the first car
index <- 1
while (counter < 8) {
    # to access the speed of the car
    speed <- cars[index, "speed"]
    # if it is more than 9
    if (speed > 9) {
        # we print the car found, with \n to go to the next line
        cat("Car #", index, "speed:", speed, "\n")
        # and track that we have printed one more car
        counter <- counter + 1
        }
    # every time we go to the next car
    index <- index + 1
    }
```

```
Car # 7 speed: 10
Car # 8 speed: 10
Car # 9 speed: 10
Car # 10 speed: 11
Car # 11 speed: 11
Car # 12 speed: 12
Car # 13 speed: 12
Car # 14 speed: 12
```

One way to do it with what we learned

- How many cars have a speed greater than 10? What is their mean distance?

```
# we put 0 to add up the values we have while browsing the cars
count    <- 0
distance <- 0
# we browse all cars by their index
for (i in 1:nrow(cars)) {
    # test if the speed of the car exceeds 10
    if (cars[i, "speed"] > 10) {
        # here we add it to the group of cars considered
        count    <- count + 1
        # we add its distance to compute the mean afterwards
        distance <- distance + cars[i, "dist"]
        }
    }
# we compute the distance with the global sum of all distances and the number
of # cars used to get that global sum
distanceMean <- distance / count
# print the results, “\n” is used to go to the next line
cat(count, "cars, mean distance", distanceMean, "\n")
```

```
41 cars, mean distance 48.95122
```

Creating functions

To create a function you need to:

- State you want to create a function with *function()*
- Include required arguments in brackets *()*
- Contain the commands in the curly brackets *{}*
- State your return object, using *return()*

```
> function.example <- function(vector.of.values){
+ sum.exponent.value <- sum(vector.of.values)^2
+ return(sum.exponent.value)
+ }

> dataset.a
[1]  1 22  3  4  5
> function.example(dataset.a)
[1] 1225
```

Creating functions continued…

- You can add in default values to arguments

```
> function.example <- function(vector.of.values, exponent.value = 2){
+ sum.exponent.value <- sum(vector.of.values)^exponent.value
+ return(sum.exponent.value)
+ }

> dataset.a
[1]  1 22  3  4  5
> function.example(dataset.a)
[1] 1225
> function.example(dataset.a, exponent.value = 10)
[1] 2.758547e+15
```

et’s try! (VI)

- Create a function that takes in a vector and returns its mean
- Create a function that takes in a numeric vector and minimum cutoff value. Return the mean, median and variance for the numbers in the vector that are greater than the minimum cutoff. Use all positive values if the user does not input a minimum cutoff value

Possible Solutions

- Create a function that takes in a vector and returns its mean

```
> calculate.mean <- function(x){
+ to.return <- mean(x)
+ return(to.return)
+ }

> dataset.a
[1]  1 22  3  4  5
> calculate.mean(dataset.a)
[1] 7
```

Possible Solutions

- Create a function that takes in a numeric vector and minimum cutoff value. Return the mean, median and variance for the numbers in the vector that are greater than the minimum cutoff. Use all positive values if the user does not input a minimum cutoff value

```
> summary.selection <- function(vector.of.values, cutoff.value = 0){
+ selected <- vector.of.values[vector.of.values > cutoff.value]
+ mean.value <- mean(selected)
+ median.value <- median(selected)
+ var.value <- var(selected)
+ to.return <- list( mean = mean.value, median = median.value, var =
var.value)
+ return(to.return)
+ }
```

```
> summary.selection(dataset.a)
$mean
[1] 7
$median
[1] 4
$var
[1] 72.5
```