R - Getting Started

R Packages

Get add-on packages

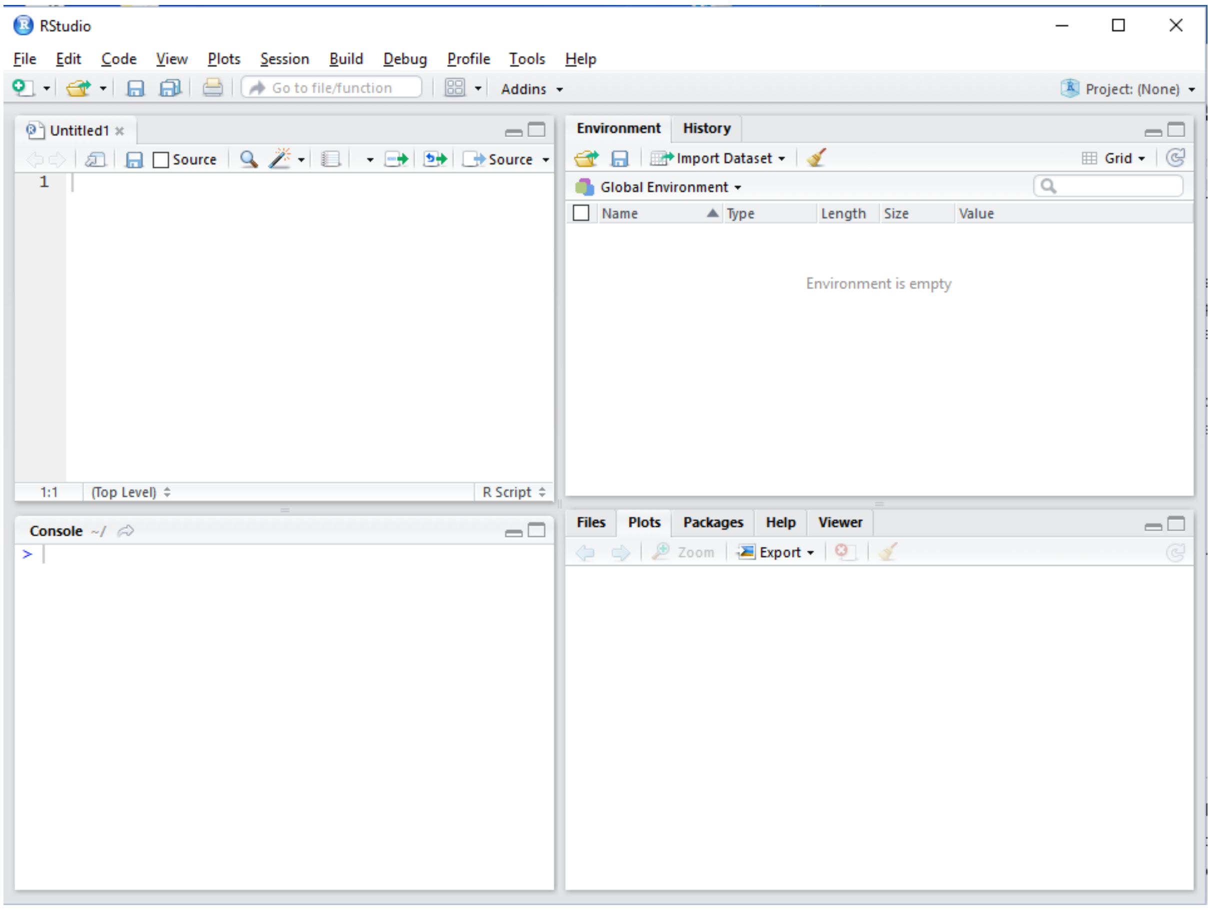
R has become very popular due its collection of user-contributed packages. There are over 5,000 packages available often written by authorities in the field of statistics and data mining.

A package is essentially a library of prewritten code to accomplish some task or set of tasks. For example, the ggplot2 package is used for plotting and sp is used for dealing with spatial data. Not all packages are written to the same quality and robustness and were also written for statisticians and not software engineers.

R has a core set of command libraries (**base**, **graphics**, **stats**, etc), but there is a wealth of add-on packages available (the full list is available at the CRAN web site).

RStudio

While there are a number of IDEs available, the best right now is RStudio and is freely available. It is available for Windows, MAC and Linux. RStudio is highly customisable but the basic interface looks like this.



In this case the lower left pane is the R Console, which can be used just like the standard R console. The upper left pane takes the place of a text editor but is far more powerful, The upper right pane holds information about the workspace, command history, files in the current folder. The lower right pane displays plots, package information and help files.

Packages already included

The following are a few of the add-on packages already included with your standard R installation.

boot –bootstrap resampling
foreign – read dainstata from files in the format of other stats programs
lattice– multi-panel graphics
mgcv – generalized additive models
nlme– linear mixed-effects models, generalized least squares

Example packages available for download

Most R packages are **not** included with the standard installation, and you need to download and install it before you can use it. The full list of available packages is here.
To install a package from the command line, execute the following command

```
install.packages("ggplot2", dependencies=TRUE)
```

The packages are installed and they are almost ready to use and just need to be loaded first.

```
library(packagename)     OR require(packagename)
# e.g.
library(ggplot2)
```

A package only needs to be loaded when a new R session is started. Once loaded, it remains available until R is restarted or the package is unloaded. You must do this again every time you run R.

To unload a package:

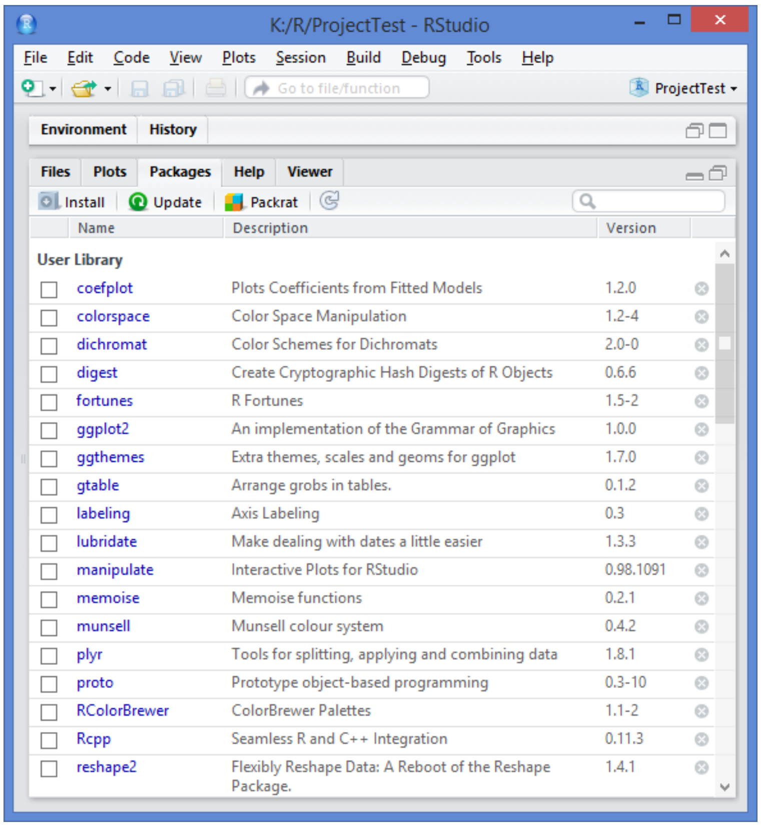
```
detach("package:ggplot2")
```

OR alternatively, uncheck the checkbox next to the package name in RStudio Packages pane

To see all the libraries available on your computer enter

```
library()
```

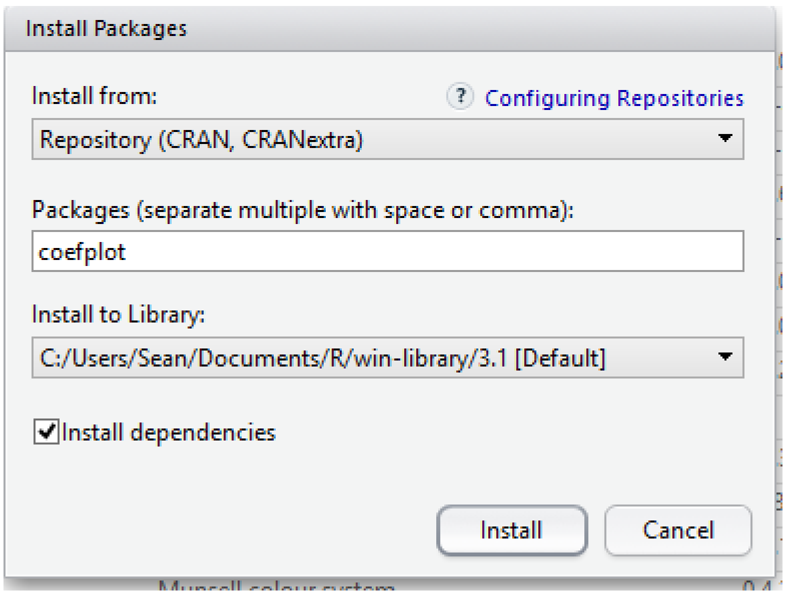
Note this can also be done by clicking the install button under the packages tab in the bottom right pane in RStudio. When installed and you want to use this package, click the checkbox associated with the package. In RStudio, access the packages pane by either clicking its tab or pressing Ctrl+7 on the keyboard. Support is also provided in the dropdown menus Tools>Install Packages


K:/R/ProjectTest - RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
ProjectTest
Environment History
Files Plots Packages Help Viewer
Install Update Packrat

User Library

Name	Description	Version
coefplot	Plots Coefficients from Fitted Models	1.2.0
colorspace	Color Space Manipulation	1.2-4
dichromat	Color Schemes for Dichromats	2.0-0
digest	Create Cryptographic Hash Digests of R Objects	0.6.6
fortunes	R Fortunes	1.5-2
ggplot2	An implementation of the Grammar of Graphics	1.0.0
ggthemes	Extra themes, scales and geoms for ggplot	1.7.0
gtable	Arrange grobs in tables.	0.1.2
labeling	Axis Labeling	0.3
lubridate	Make dealing with dates a little easier	1.3.3
manipulate	Interactive Plots for RStudio	0.98.1091
memoise	Memoise functions	0.2.1
munsell	Munsell colour system	0.4.2
plyr	Tools for splitting, applying and combining data	1.8.1
proto	Prototype object-based programming	0.3-10
RColorBrewer	ColorBrewer Palettes	1.1-2
Rcpp	Seamless R and C++ Integration	0.11.3
reshape2	Flexibly Reshape Data: A Reboot of the Reshape Package.	1.4.1

Click Install to install a package. Then type the name,(note multiple packages can be installed separated by commas.), select the install dependencies checkeboxs. This will automatically download and install all packages that the desired package requires to work.



Install Packages

Install from: Repository (CRAN, CRANextra)

Packages (separate multiple with space or comma): coefplot

Install to Library: C:/Users/Sean/Documents/R/win-library/3.1 [Default]

[✓] Install dependencies

Uninstalling packages can done from the install window in RStudio or by using the remove.packages where the first argument is a character vector naming the packages to be removed.

```
remove.packages("coefplot")
```

Getting help

Built-in help

Use "?" in the R command window to get documentation of specific command. For example, to get help on the "mean" function to calculate a sample mean, enter

```
?mean
```

You can also search the help documentation on a more general topic using "??" or "help.search" or find a list using "apropos" . For example, use the following commands to find out what's available on kmeans and k-nearest neighbour models.

```
apropos("mean")

??knearest

??"kmeans"   # same as help.search("kmeans")
```

A window will display and lists commands available and the packages that include them. Note the "??" command will only search documentation in the R packages installed on your computer.

Interpreting a help page

As an example, here's how to interpret the help page for the sample mean, obtained by

```
?mean
```

In the pop-up help window, look under the title "Usage" and you will see something like this:

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

The items between the brackets "()" are called **arguments**.
Any argument without an "=" sign is **required** — you must provide it for the command to work. Any argument with an "=" sign represents an **option**, with the default value indicated. (Ignore the "…" for now.)
In this example, the argument "x" represents the data object you supply to the function. Look under "Arguments" on the help page to see what kind of object R needs. In the case of the mean almost any data object will do, but you will usually apply the function to a vector (representing a single variable). If you are happy with the default settings, then you can use the command in its simplest form. If you want the mean of the elements in the variable "myvariable", enter

```
myvariable <- c(1,2.22,3.1,4.333,NS,5.0,6,7.5,8,9,10)
mean(myvariable)
```

If the default values for the options don't meet your needs you can alter the values. The following example changes the "na.rm" option to TRUE. This instructs R to remove missing values from the data object before calculating the mean. (If you fail to do this and have missing values, R will return "NA".)

```
mean(myvariable)
mean(myvariable, na.rm=TRUE)
```

The following example changes the "trim" option to calculate a trimmed mean,

```
mean(myvariable, na.rm=TRUE, trim=0.1)
```

Online help

R commands to analyze the data for all examples presented in the 2nd edition of *The Analysis of Biological Data* by Michael Whitlock and Dolph Schluter are *here*.

Tom Short's *R reference card*

Venables and Smith's *Introduction to R* (pdf file — right-click and save to disk)
An R blog! Daily news and tutorials about R.

Basic Maths

Introduction

The Hello World of R!:

Here are some examples. Try them out!

```
1+1

3*5*7

10/3

4*(6+8)
```

Variables and assignment and removal of variables

There are a number of ways to assign a value to a variable. The valid assignment operators are <- and – with the first being the preferred. Note variable names are case-sensitive.

Here are some examples

```
x <- 4
```

```
x
OUTPUT [1] 4

Y = 5
Y
OUTPUT [1] 5

z <- v <- (3*8)/2
z
OUTPUT [1] 12
rm(v)
v
OUTPUT Error: object 'v' not found
```

Datatypes

There are a numerous data types in R that store various kinds of data. The four main types of data most likely to be used are numeric, character (string), Date and logical (True/FALSE)

Numeric Data

Here are some examples:

# numeric is similar to float or double in other languages

```
# numeric is similar to float or double in other languages
x<- 4
class (x)
OUTPUT [1] numeric
is.numeric(x)
OUTPUT [1] TRUE
# x is not treated as an integer
is.integer(x)
OUTPUT [1] FALSE

#set x as an integer
OUTPUT x <-4L
is.integer(x)
OUTPUT [1] TRUE
# Note it will also the numeric test
```

- Try these out

  4L
  4L*1.3
  5L/2L

What datatype are the results? Try out the class() function

Character Data, Dates and Logical

The character string data type is very common in statistical analysis and must be handled with care. There are two primary ways of handling character data: character and factor. While they look the same, they are treated quite differently

```
x <- "mystuff"
x
OUTPUT [1] "mystuff"

y <-factor("mystuff")
y
OUTPUT [1] mystuff
Levels: mystuff
```

Note that x contains “mystuff” in quotes while y has the word “mystuff” without quotes and a second line of information about the levels of y. Factors are a special type of vector to work with categories (categorical data) e.g. a product can be red , green, black or blue. Factors are closely related to characters because any character vector can be represented by a factor. However, they are neither character vectors nor numeric vectors, although they have some attributes of both. Factors behave a little bit like character vectors in that the unique categories often are text. Factors also behave a little bit like integer vectors as R encodes the levels as integers. Factors are useful for categorical and ordinal variables

```
# create a character vector
directions <- c("north","south", "east", "south")
directions
OUTPUT [1] "north" "south" "east"  "south"

# create a factor vector
directions.factors <- factor(directions)
directions.factors
OUTPUT [1] north south east  south
Levels: east north south

#Creating an Ordinal variable
food  <-  factor(c("low",  "high",  "medium",  "high",  "low",  "medium",
"high"))
levels(food)
[1] "high"   "low"    "medium"

food <- factor(food, levels = c("low", "medium", "high"), ordered=TRUE)
min(food)
[1] low
Levels: low < medium < high

# to convert a factor to a character vector
as.character(directions.factors)
OUTPUT [1] "north" "south" "east"  "south"

# to convert a factor to a numeric vector
as.numeric(directions.factors)
OUTPUT [1] 2 3 1 3
```

Note also that characters are case-sensitive. To find the length of a character or numeric use the nchar function.

```
nchar(x)
OUTPUT [1] 7
nchar(432)
OUTPUT [1] 3
```

The most useful types for dates are DATE and POSIXct

```
date1 <- as.Date("2015-02-23")
date1
OUTPUT [1] "2015-02-23"



date2 <- as.POSIXct("2015-02-23 14:34:00")
date2
OUTPUT [1] "2015-02-23 14:34:00 GMT"

class(date1)
OUTPUT [1] "Date"
class(date2)
OUTPUT [1] "POSIXct" "POSIXt"
```

You can use function as.numeric or as.Date

```
date1 <-as.numeric(date1)
class(date1)
OUTPUT [1] "numeric"
```

Logicals are a way of representing data that be either be TRUE or FALSE. TRUE is the same as 1 and FALSE is the same as 0.

```
> TRUE+5
OUTPUT [1] 6
```

```
> TRUE+5
OUTPUT [1] 6
# R provides T and F as shortcuts for TRUE and FALSE but is best practice
not to use them.
k <- TRUE
y <-T
y
OUTPUT [1] TRUE
```

```
k
OUTPUT [1] TRUE

class(k)
OUTPUT [1] "logical"

is.logical(k)
OUTPUT [1] TRUE
```

```
#logicals can result from comparing 2 numbers or characters
2 == 3
OUTPUT [1] FALSE

2 != 3
OUTPUT [1] TRUE

2 <= 3
OUTPUT [1] TRUE

ans <- 2 < 3
ans
OUTPUT [1] TRUE
```

Getting Started - Part 2

Introduction to Vectors

A vector is a simple array of numbers or characters, such as the measurements of a single variable on a sample of individuals.

Enter values

Use the left arrow "<-" ("less than" sign followed by a dash) and the `c` function (for concatenate) to create a vector containing a set of measurements.

```
x <- c(11,42,-3,14,5)            # store these 5 numbers in vector x

x <- c(1:10)                     # store integers 1 to 10

x <- c("alias","smith","jones")    # use quotes for character data
```

Use the `seq` function to generate a sequence of numbers and store in a vector,

```
x <- seq(0,10,by=0.1)    # 0, 0.1, 0.2, ... 9.9, 10
```

(note: "seq" results that include decimals may not be exact — the result "0.2" may not be exactly equal to the number 0.2 unless rounded using the "round" command)

Use `rep` to repeat values a specified number of times and store to a vector,

```
x <- rep(c(1,2,3),c(2,1,4))         # 1 1 2 3 3 3 3
```

```
#shortcuts in creating vectors
a <- 1:10
a
OUTPUT [1]  1  2  3  4  5  6  7  8  9 10

a <- -2:3
a
OUTPUT [1] -2 -1  0  1  2  3

length(a)
OUTPUT [1] 6
```

If we have 2 vectors of equal length, each of the corresponding elements can be operated on together e.g.

```
a <- -2:3
x <- 1:10
y <- -5:4
x-y
OUTPUT [1] 6 6 6 6 6 6 6 6 6 6
```

```
a-x              # will throw an error

x*y
OUTPUT [1] -5 -8 -9 -8 -5  0  7 16 27 40

# note dividing can result  Inf   (infinity)
 x/y
OUTPUT [1] -0.2 -0.5 -1.0 -2.0 -5.0  Inf  7.0  4.0  3.0  2.5

#raise one to the power of the other
x^y
OUTPUT   [1]   1.000000e+00   6.250000e-02   3.703704e-02   6.250000e-02
2.000000e-01
[6] 1.000000e+00 7.000000e+00 6.400000e+01 7.290000e+02 1.000000e+04

x <5
OUTPUT [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE

#  to  test  whether  all  the  resulting  elements  are  TRUE  use  the  all
function,
all(x<y)
OUTPUT [1] FALSE

# to test whether any element in the resulting elements are TRUE use the
any   function,
any(x<y)
OUTPUT [1] FALSE

#creating a character vector and check the length of each value
z <- c("dublin", "offaly", "kildare")
nchar(z)
OUTPUT [1] 6 6 7

#accessing  individual  elements  of  a  vector  is  done  using  square
brackets([])
# 1st position in vector
z[1]
OUTPUT [1] "dublin"

# 1st  to 2nd  elements in vector
z[1:2]
OUTPUT [1] "dublin" "offaly"

# non-consecutive elements
z[c(1,3)]
OUTPUT [1] "dublin"  "kildare"
```

```
# Providing names for each element
z <- c("champs!"="dublin1", "chimps!"="mayo", "chumps!"="kerry!")
z
OUTPUT  Champs!   chimps!   chumps!
        "dublin"  "offaly" "kildare"


w <- 1:3
names(w) <- c("a","b","c")
w
OUTPUT a b c
       1 2 3
```

Calling functions

Some simple examples:

```
x <- c(1,2,3,1,4,23,6,7,7,7,9,8,5)
median(x)
[1] 6
mean(x)
[1] 6.384615
```

Missing Data

Missing Data play a critical role in both statistics and computing, and R has two types of missing data, NA and NULL.

NA

We can have missing data values for many reasons. Common techniques to represent missing data are a dash, a period or the number 99. R uses NA e.g.

```
z <- c(1,2,3,NA,5,NA,-99)
is.na(z)
OUTPUT [1] FALSE FALSE FALSE  TRUE FALSE  TRUE

z[5] <- NA         # assign "missing" to the 5th element of x

z[z == -99] <- NA # change all instances of -99 in x to missing

which(is.na(z))    # identify which element(s) is missing

x1 <- na.omit(z)   # put the non-missing values of x into new vector x1

x1 <- z[complete.cases(z)] # same concept

x1 <- z[!is.na(z)]        # same concept!
```

```
length(x1)                   # count the number of non-missing values
```

NULL

NULL is the absence of a value i.e. it is not exactly missingness, it is nothingness. Functions can sometimes return a NULL and their arguments can be NULL. An important difference between NA and NULL is that NULL is atomical and cannot exist within a vector. If used inside a vector it simply disappears.

```
z <- c(1,2,3,NULL,5,NULL)
z
OUTPUT [1] 1 2 3 5

d <- NULL
e <- 4
is.null(d)
OUTPUT [1] TRUE

is.null(e)
OUTPUT [1] FALSE
```

The test for NULL is is.null. Since NULL cannot be part of a vector, is.null is appropriately not vectorised.

More on Vectors

Delete a vector

The following command removes the vector x from the local R environment.

```
rm(x)
```

Access elements of a vector

Use integers in square brackets indicate a predicate specifying the elements of a vector required. For example,

```
x <- c(1:10)   # create a vector x with values 1 to 10

x[5]           # 5th value of the vector x

x[2:6]         # 2nd through 6th elements of x

x[2:length(x)] # everything but the first element

x[-4]          # everything but the fourth element

x[5] <- 4.2    # change the value of the 5th element to 4.2
```

A little more on Vectors

Maths with vectors

These operations are carried out on every element of the vector

```
#Create a vector
x <- c(1:10)
x + 1          # add 1 to each element of x
x^2            # square each element of x
x/2            # divide each element of x by 2
10 * x         # multiply each element of x by 10
```

Operations on two vectors x and y work best when both are the same length (have the same number of elements). For example

```
y <- c(10:1)

x * y        # yields a new vector whose

             # elements are x[1]*y[1], x[2]*y[2], ... x[n]*y[n]
```

Note: If x and y are not the same length, then the shorter vector is elongated by starting again at the beginning.

Useful vector functions

Here is a selection of useful functions for data vectors. Many of the functions will also work on other data objects such as data frames, possibly with different effects.

Transform numerical data

The most common data transformations, illustrated using the single variable “x”.

```
sqrt(x)          # square root
sqrt(x+0.5)      # modified square root transformation
log(x)           # the natural log of x
log10(x)         # log base 10 of x
exp(x)           # exponential ("antilog") of x
abs(x)           # absolute value of x
asin(sqrt(x))    # arcsine square root (used for proportions)
```

Statistics

Here are a few basic statistical functions on a numeric vector named x. **Most of them will require the na.rm=TRUE option if the vector includes one or more missing values**.

```
x <- c(1:10)
sum(x)                 # the sum of values in x
length(x)              # number of elements (including missing)
mean(x)                # arithmetic mean
var(x)                 # sample variance
sd(x)                  # sample standard deviation
min(x)                 # smallest element in x
max(x)                 # largest element in x
range(x)               # smallest and largest elements in x
median(x)              # median of elements in x
quantile(x)            # quantiles of x
unique(x)              # extracts only the unique values of x
sort(x)                # sort, smallest to largest

# weights to be applied to  vector x
w <- c(2.3,3.4,3.2,5.6,1.2,4.5,1.2,3.5,1.7,2.6)
weighted.mean(x, w)    # weighted mean
```

? do you agree with the result. Write some R statements to verify the result from the weighted.mean function

Sol

```
> w.sum <- sum(w)
> w.sum
[1] 29.2
> sum(w*x)
[1] 151.8
> sum(w*x)/w.sum
[1] 5.19863
```

Functions for character data

```
x <- c("Philip","Paula", "Mary", "John", "Paul")
casefold(x)              # convert to lower case
casefold(x,upper=TRUE)   # convert to upper case
substr(x,2,4)       #extract 2nd to 4th characters of each element of x
paste(x,"zz",sep="")     # paste "ly" to the end of each element in x
nchar(x)                 # no. of characters in each element of x
grep("ry",x)              # which elements of x contain letter "a" ?
strsplit(x,"a")     # split x into pieces wherever the letter "a" occurs
```

Exercises

1. Create the following vectors

   q = [-1.2, 2, 3, -4.3, 3, 4, 5, 6, -5, -1.2] and

   r = 10.2, -23, 4.3, -6.6 , 13, 42, 15, -16, 5, -10.2]

2. Find the length of the vectors. If they are not equal, using subscripting add more values accordingly.
3. For q, Provide the "5 number (statistical) summary" for each of the vectors . Sort the vector and check do you agree with the answers? What is the Interquartile Range?
4. What vector has the largest sum.? Using absolute values what vector has the largest sum? Compare the means and medians and standard deviations
5. Create the character vector "Philip","Paula", "Mary", "John", "Paul" .
   - Output the positions that contain "au"
   - How many values contain "au"
   - Output the values that contain "au"
   - Output the length of each of the strings that contain "au"

```
•
   1
1.
q <- c(-1.2, 2,3, -4.3, 3, 4, 5, 6, -5, -1.2)
r<-c(10.2, -23, 4.3, -6.6 , 13, 42, 15, -16, 5)

2. length(r) etc
   r[10] <- -10.2

3.
min(q)
max(q)
median(q)
quantile(q,1/4)
quantile(q,3/4)
quantile(q,3/4) - quantile(q,1/4) or IQR(q)

summary(q)
4.

sumq <- sum(q)
sumr <- sum(r)

sumq <- sum(abs(q))
mean(q)
sd(q)


5.

x<- c("Philip","Paula", "Mary", "John", "Paul")
```

```
z<-grep("au",x)
z
length(grep("au",x)) #the number of strings that contain au

x[z]
nchar(x[z])
```