

# CB2-101: R introduction

*Malay (malay@uab.edu)*

*November 6, 2017*

## Contents

<b>1</b>	<b>History of R</b>	<b>2</b>
<b>2</b>	<b>Starting R</b>	<b>2</b>
<b>3</b>	<b>R environment</b>	<b>2</b>
3.1	Exercise . . . . .	3
<b>4</b>	<b>Getting help</b>	<b>3</b>
4.1	Some useful online documentation . . . . .	3
4.2	Finding help inside R . . . . .	3
4.3	R for bioinformatics . . . . .	3
<b>5</b>	<b>Installing packages in R</b>	<b>4</b>
<b>6</b>	<b>Assignment</b>	<b>4</b>
<b>7</b>	<b>Basic data types in R</b>	<b>4</b>
<b>8</b>	<b>R data structures</b>	<b>4</b>
8.1	Vectors . . . . .	4
8.1.1	Creating vectors . . . . .	4
8.1.2	Accessing elements of a vector . . . . .	4
8.1.3	Modify vector . . . . .	5
8.1.4	Converting vector of one type to another . . . . .	5
8.2	Lists . . . . .	5
8.2.1	Creating list . . . . .	6
8.2.2	Access elements of a list . . . . .	6
8.2.3	Modify lists . . . . .	6
8.3	Matrix . . . . .	6
8.4	Factor . . . . .	7
8.5	Missing values . . . . .	8
8.6	Data frames . . . . .	8
8.6.1	Reading and writing data-frame . . . . .	8
<b>9</b>	<b>Names</b>	<b>9</b>
<b>10</b>	<b>Subsetting</b>	<b>10</b>
10.1	Subsetting vector . . . . .	10
10.2	Subsetting list . . . . .	10
10.3	Removing missing values . . . . .	11
10.4	Matrix subsetting . . . . .	11
10.5	Subsetting data-frame . . . . .	11
<b>11</b>	<b>Vectorized operations</b>	<b>12</b>

<b>12 Some programming concepts in R</b>	<b>13</b>
12.1 Functions . . . . .	13
12.2 For loop and if condition . . . . .	13
12.3 Reading a file line by line in R . . . . .	13
12.4 Regular expression in R . . . . .	13
<b>13 Summary statistics</b>	<b>14</b>
<b>14 Random numbers</b>	<b>14</b>
<b>15 Plotting in R</b>	<b>15</b>
15.1 Plot . . . . .	15
15.1.1 Building a plot from pieces . . . . .	17
15.1.2 Plotting characters . . . . .	17
15.2 Drawing devices in R . . . . .	17
15.3 <code>par</code> . . . . .	18
15.4 Drawing multiplots . . . . .	18
15.5 Histograms . . . . .	18
15.6 Boxplots . . . . .	19
15.7 Barplots . . . . .	20
15.8 Barplots with error bars . . . . .	22
<b>16 Colors in R</b>	<b>23</b>
<b>17 Linear regression and correlation</b>	<b>25</b>
17.1 Linear regression . . . . .	25
17.2 Correlations . . . . .	26

## 1 History of R

R is the *lingua franca* of statistical software. It is a dialect of S, a computer language developed by John Chambers in 1976 at Bell Labs. In 1988, the system was rewritten in C. The software changed hand frequently, until in 1993, R appeared in the scene. It was a *free* implementation of S, written by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand.

## 2 Starting R

If you're in command-line environment you can start R by typing the command `R`. In our example will use a far nicer environment called RStudio. It is available from: <https://www.rstudio.com/>.

## 3 R environment

1. `getwd()` Get the current directory
2. `setwd()` Set the current directory
3. `ls()` Lists the objects created.
4. `rm()` Will remove objects.
5. `list.files()` or `dir()` List the files in the current directory
6. `source()` Reads a R file

### 3.1 Exercise

Create a file called `source_test.R` to have the following code.

```
myfunction <- function () {  
  x <- rnorm(100)  
  mean(x)  
}  
  
myfunction()
```

```
## [1] 0.07146841
```

Type `source("source_test.R")` in the R console. Type `ls()`, you'll see the object `myfunction` has been created in the workspace. You can also run the script from the command line.

```
Rscript source_test.R
```

```
## [1] -0.003655489
```

## 4 Getting help

1. CRAN - The Comprehensive R Archive Network (<http://cran.r-project.org/>). About ~4000 packages and data available for use.
2. R website (<http://www.r-project.org/>) and FAQs (<http://cran.r-project.org/doc/FAQ/R-FAQ.html>)
3. R related projects are difficult to search using Google. Use R-specific search engine RSEEK (<http://www.rseek.org/>).
4. A very good guide to R is "R Inferno" ([http://www.burns-stat.com/pages/Tutor/R\\_inferno.pdf](http://www.burns-stat.com/pages/Tutor/R_inferno.pdf)).
5. Hadley Wickham's "Advanced R" (<http://adv-r.had.co.nz/>).

### 4.1 Some useful online documentation

1. An introduction to R (<http://cran.r-project.org/doc/manuals/r-release/R-intro.pdf>)
2. R Data Import/Export (<http://cran.r-project.org/doc/manuals/r-release/R-data.pdf>)

A full list of such manuals can be found at <http://cran.r-project.org/manuals.html>.

### 4.2 Finding help inside R

1. `help(object)` will show the help page of any object including function in R.
2. `?object` is same thing as above.
3. `help.search("string")` will search for a string pattern in all help files.
4. `??string` is the same thing as above
5. `help(package="packagename")` will show the start help page of a package.

### 4.3 R for bioinformatics

1. **Bioconductor project** <http://www.bioconductor.org/> A set of R modules specifically meant for biological data analysis. Distributed separately from CRAN.
2. Some biological data related module in CRAN, such as **ape**, a phylogenetic analysis package. Or, **seqinr**, a basic sequence analysis package. A brief introduction to using R for bioinformatics can be found [here](#).

## 5 Installing packages in R

`install.packages('Packagename')` will install any package in R. To use the package you need to use `library('Packagename')`.

## 6 Assignment

Assignment operator in R is `<-`.

```
x<- 1:20
x
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

## 7 Basic data types in R

1. character
2. numeric All numbers in R are double precision real numbers. If you want an integer, you need to specify L suffix. 1 is `numeric`, but 1L is integer.
3. integer
4. complex
5. logical (True/False)

Undefined data in R is represented as either 'NaN' or 'NA' types. Infinity is represented as `Inf`.

## 8 R data structures

### 8.1 Vectors

1. Basic data structures in R.
2. All elements are of same type.
3. `typeof()` returns data-type.
4. `length()` returns the number of elements.

#### 8.1.1 Creating vectors

```
x <- vector()      # Empty vector
x <- c(1,2,3,4)     # Using c()
x <- 1:10           # Using ":" operator
x <- seq(1,10,by=1) # Using seq() function
```

#### 8.1.2 Accessing elements of a vector

Elements of vector in R starts with 1, rather than 0 like in other languages.

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

```
x[3]      # Access the 3rd element

## [1] 3
x[-1]     # Access everything except the first element

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

### 8.1.3 Modify vector

```
x

## [1] 1 2 3 4 5 6 7 8 9 10
x[2] <- 0; x      # Modify 2nd element

## [1] 1 0 3 4 5 6 7 8 9 10
x[x > 5] <- 1; x  # Modify elements greater than 5

## [1] 1 0 3 4 5 1 1 1 1 1
x <- c(1,2); y <- c(2,4)
z <- c(x,y)       # Joining two vectors
z

## [1] 1 2 2 4
```

### 8.1.4 Converting vector of one type to another

Use `as.x()` funtions.

```
x <- 0:6
as.numeric(x)

## [1] 0 1 2 3 4 5 6
as.logical(x)

## [1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE
as.character(x)

## [1] "0" "1" "2" "3" "4" "5" "6"
as.complex(x)

## [1] 0+0i 1+0i 2+0i 3+0i 4+0i 5+0i 6+0i
```

## 8.2 Lists

1. Contains mixed data type.
2. `typeof()` returns “list”.
3. `length()` returns length.

### 8.2.1 Creating list

```
x <- list() # Empty list
x <- list("a" = 2.5, "b" = TRUE, "c" = 1:3) # Mixed type list
x
```

```
## $a
## [1] 2.5
##
## $b
## [1] TRUE
##
## $c
## [1] 1 2 3
```

```
typeof(x)
```

```
## [1] "list"
```

```
length(x)
```

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

### 8.2.2 Access elements of a list

```
x$a      # Access using name
```

```
## [1] 2.5
```

```
x[c(1:2)] # Using index
```

```
## $a
## [1] 2.5
##
## $b
## [1] TRUE
```

```
x[["b"]] # Using name
```

```
## [1] TRUE
```

### 8.2.3 Modify lists

```
x[["d"]] <- 25
```

## 8.3 Matrix

Special type of vector with “dimension” attributes.

```
m<-matrix(nrow=2, ncol=3)
dim(m)
```

```
## [1] 2 3
```

```
attributes(m)
```

```
## $dim  
## [1] 2 3
```

Matrix is constructed column-wise.

```
m<-matrix(1:6, nrow=2, ncol=3)  
m
```

```
##      [,1] [,2] [,3]  
## [1,]    1    3    5  
## [2,]    2    4    6
```

```
m <- 1:10  
dim(m)<-c(2,5)  
m
```

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

You can create matrix out of vectors using column-binding or row-binding:

```
x<- 1:3  
y <- 10:12  
cbind(x,y)
```

```
##      x  y  
## [1,] 1 10  
## [2,] 2 11  
## [3,] 3 12
```

```
rbind(x,y)
```

```
##      [,1] [,2] [,3]  
## x      1    2    3  
## y     10   11   12
```

## 8.4 Factor

Categorical data. Can be ordered or unordered.

```
x<- factor (c("yes","yes", "no"))
```

Factors have levels:

```
table(x)
```

```
## x  
## no yes  
##  1  2
```

Factors are basically numerical data under the hood. Each string is given a number. We can view the underlying assignment using this:

```
unclass(x)
```

```
## [1] 2 2 1  
## attr("levels")
```

```
## [1] "no" "yes"
```

Explicit ordering can be specified using `levels` arguments to `factor`. If we want to make `no` before `yes`:

```
x<- factor (c("yes", "no"), levels=c("Yes", "no"))
```

## 8.5 Missing values

Undefined values in R are represented by `"NaN"` and `"NA"`. NA values can have class of numeric, integer, etc. NaN is also NA but the converse is not true. There are two functions to find NaN and NA in R:

```
is.na()
is.nan()
```

```
x<- c(1,2, NA, 10, 3)
is.na(x)
```

```
## [1] FALSE FALSE TRUE FALSE FALSE
```

```
is.nan(x)
```

```
## [1] FALSE FALSE FALSE FALSE FALSE
```

```
x<- c(1,2,NaN,NA,4)
is.na(x)
```

```
## [1] FALSE FALSE TRUE TRUE FALSE
```

```
is.nan(x)
```

```
## [1] FALSE FALSE TRUE FALSE FALSE
```

## 8.6 Data frames

Key data type in R. It stores tabular data. Column of a data frame can have different types of data, unlike matrix (same type). Special attributes `row.names`. To read use `read.table()` or `read.csv()`. Data frame can be converted to matrix `data.matrix()`.

```
x <-data.frame (foo=1:4, bar=c(T,T,F,F))
x
```

```
##   foo  bar
## 1   1 TRUE
## 2   2 TRUE
## 3   3 FALSE
## 4   4 FALSE
```

```
nrow(x)
```

```
## [1] 4
```

```
ncol(x)
```

```
## [1] 2
```

### 8.6.1 Reading and writing data-frame



```
data(iris) # Read iris data sets
head(iris) # View the first few line of the iris data
```

```
## Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1          3.5          1.4          0.2 setosa
## 2          4.9          3.0          1.4          0.2 setosa
## 3          4.7          3.2          1.3          0.2 setosa
## 4          4.6          3.1          1.5          0.2 setosa
## 5          5.0          3.6          1.4          0.2 setosa
## 6          5.4          3.9          1.7          0.4 setosa
```

```
write.table(iris,"iris.txt",quote=F) # Write a sample file out
dir() # Check that we have written a file
```

```
## [1] "airquality.pdf" "Intro_to_R.pdf" "Intro_to_R.Rmd" "iris.txt"
## [5] "source_test.R"
```

```
test_data<-read.table("iris.txt", header=T) # read the data back into test_data
head(test_data) # Check the first few lines of the data
```

```
## Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1          3.5          1.4          0.2 setosa
## 2          4.9          3.0          1.4          0.2 setosa
## 3          4.7          3.2          1.3          0.2 setosa
## 4          4.6          3.1          1.5          0.2 setosa
## 5          5.0          3.6          1.4          0.2 setosa
## 6          5.4          3.9          1.7          0.4 setosa
```

To read a data frame from a gzipped file:

```
d<-read.table(gzfile("myzipped.gz"))
```

## 9 Names

Each type of data in R can have names. But they are most important for Matrix and Data-frames.

```
m<- matrix(1:4, nrow=2, ncol=2)
dimnames(m) <- list (c("a","b"), c("c","d"))
m
```

```
## c d
## a 1 3
## b 2 4
```

```
d<-data.frame(c(1,2,3),c(4,5,6))
d
```

```
## c.1..2..3. c.4..5..6.
## 1          1          4
## 2          2          5
## 3          3          6
```

```
names(d)<- c("A","B")
d
```

```
## A B
## 1 1 4
```

```
## 2 2 5
## 3 3 6
```

## 10 Subsetting

Subsetting is a way to select a subset of data. There are 3 basic ways to subset data in R:

1. `[]`
2. `[[]]`
3. `$`

### 10.1 Subsetting vector

```
x<-c ("a","b", "c")
x[1]

## [1] "a"
x[2]

## [1] "b"
x[1:3]

## [1] "a" "b" "c"
x[-2]

## [1] "a" "c"
x[x > "a"]

## [1] "b" "c"
u <- x > "a"
u

## [1] FALSE TRUE TRUE
x[u]

## [1] "b" "c"
```

### 10.2 Subsetting list

```
x <- list (foo=1:4, bar=0.6)
x[1]

## $foo
## [1] 1 2 3 4
x[[1]]

## [1] 1 2 3 4
x$bar

## [1] 0.6
```

```
x[["bar"]]
```

```
## [1] 0.6
```

```
x["bar"]
```

```
## $bar
```

```
## [1] 0.6
```

### 10.3 Removing missing values

```
x <- c(1,2, NA, 4, NA, 5)
```

```
bad <- is.na(x)
```

```
x[!bad]
```

```
## [1] 1 2 4 5
```

### 10.4 Matrix subsetting

```
x<- matrix (1:6, 2, 3) # Create a sample 2x3 matrix
```

```
x # print the matrix out
```

```
##      [,1] [,2] [,3]
```

```
## [1,]    1    3    5
```

```
## [2,]    2    4    6
```

```
x[1,2] # what is the value in row 1, column 2
```

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

```
x[2,1] # what is the value in row 2, column 1
```

```
## [1] 2
```

We can select the whole row or columns:

```
x[1,] # select whole row 1
```

```
## [1] 1 3 5
```

```
x[,2] # select whole column 2
```

```
## [1] 3 4
```

### 10.5 Subsetting data-frame

All the operations of matrix also work with data-frame. However, you will more often use `$` to extract individual columns of a data-frame.

```
x<-data.frame(c(1:3),c(4:6)) # create a small data frame
```

```
names(x)<-c("A","B") # Give each column a nice name
```

```
x
```

```
##   A B
```

```
## 1 1 4
```

```
## 2 2 5
```

```
## 3 3 6
x$A # Extract the column A
## [1] 1 2 3
```

## 11 Vectorized operations

Instead of using loops, you should all the time try to use “vectorized” operations. That means if you are interested in operations performed on each element of a “collection”, you should use the collection as a whole, not individual elements. This gives the computer to parallelize the operation and usually results in faster runtime.

```
x<-1:4; y<-6:9
x+y
```

```
## [1] 7 9 11 13
x-y
```

```
## [1] -5 -5 -5 -5
x*y
```

```
## [1] 6 14 24 36
x/y
```

```
## [1] 0.1666667 0.2857143 0.3750000 0.4444444
```

When you multiply two matrices using the “\*”, it multiplies element by element multiplications. This is different that standard matrix multiplication. If you are interested standard matrix multiplication, use “%\*%” operator.

```
m1<-matrix(1:4,2,2)
m1
```

```
##      [,1] [,2]
## [1,] 1    3
## [2,] 2    4
```

```
m1*m1
```

```
##      [,1] [,2]
## [1,] 1    9
## [2,] 4   16
```

```
m1 %*% m1
```

```
##      [,1] [,2]
## [1,] 7   15
## [2,] 10  22
```

## 12 Some programming concepts in R

### 12.1 Functions

```
myfunction<-function() {  
  cat("Hello world\n");  
}
```

### 12.2 For loop and if condition

```
for (i in 1:3) {  
  if (i > 1) {  
    cat("I said hello", i, "times\n")  
  }else {  
    cat("I said hello", i, "time\n")  
  }  
}
```

```
## I said hello 1 time  
## I said hello 2 times  
## I said hello 3 times
```

### 12.3 Reading a file line by line in R

```
fh <- file( inputfile, open="r" )  
while (length( line <- readLines( fh, n=1, warn= FALSE ) ) > 0) {  
  # Do something with the line  
}  
close(fh)
```

### 12.4 Regular expression in R

Some useful functions that can be used with regular expression:

1. `grep()` finds a pattern in a vector or character.
2. `sub()` substitute a text for the first occurrence.
3. `gsub()` substitute every occurrence.
4. `regexpr()` find and extracts values using a pattern.
5. `regexec()` find and extracts values. Useful for the function `regmatches()`

```
text <- "gi|123456|ref|ABCDEFG" # Text to search  
p <- "gi\\|(\\d+)\\|ref\\|(\\S+)" # Extract gi and acc  
m <- regexec(p, text, perl = TRUE) # Find match and extract  
s <- regmatches(text, m) # Extract the substring  
s[[1]][2] # Print GI
```

```
## [1] "123456"
```

```
s[[1]][3] # Print accession
```

```
## [1] "ABCDEFG"
```

## 13 Summary statistics

```
x<-rnorm(50)
mean(x)

## [1] 0.1017831

sd(x)

## [1] 0.9839159

var(x)

## [1] 0.9680904

median(x)

## [1] 0.2025304

sum(x)
```

```
## [1] 5.089156
```

You need to skip NA for doing summary statistics:

```
data(airquality)
mean(airquality$Ozone)

## [1] NA

mean(airquality$Ozone,na.rm=T)

## [1] 42.12931
```

summary() will give you a nice summary of data:

```
summary(airquality)
```

##	Ozone	Solar.R	Wind	Temp
##	Min. : 1.00	Min. : 7.0	Min. : 1.700	Min. :56.00
##	1st Qu.: 18.00	1st Qu.:115.8	1st Qu.: 7.400	1st Qu.:72.00
##	Median : 31.50	Median :205.0	Median : 9.700	Median :79.00
##	Mean : 42.13	Mean :185.9	Mean : 9.958	Mean :77.88
##	3rd Qu.: 63.25	3rd Qu.:258.8	3rd Qu.:11.500	3rd Qu.:85.00
##	Max. :168.00	Max. :334.0	Max. :20.700	Max. :97.00
##	NA's :37	NA's :7		
##	Month	Day		
##	Min. :5.000	Min. : 1.0		
##	1st Qu.:6.000	1st Qu.: 8.0		
##	Median :7.000	Median :16.0		
##	Mean :6.993	Mean :15.8		
##	3rd Qu.:8.000	3rd Qu.:23.0		
##	Max. :9.000	Max. :31.0		
##				

## 14 Random numbers

We can generate random numbers from the normal distribution using `rnorm()` function.

```
x<- rnorm(10)
x
```

```
## [1] 0.67578697 0.09123520 -0.09472224 -0.14945154 0.47859973
## [6] -0.33142486 1.57947364 0.96393415 2.45115608 -0.35481495
```

If we are interested in generating random numbers using a particular mean and standard deviation, we should do this:

```
x<-rnorm(10, mean=5, sd=2)
x
```

```
## [1] 8.659294 2.241743 4.766997 3.266976 5.738876 4.551491 3.650665
## [8] 6.797511 2.796984 3.848422
```

In R, you can sample other distributions, I will leave it up to you to explore.

## 15 Plotting in R

There are 3 graphical libraries in R:

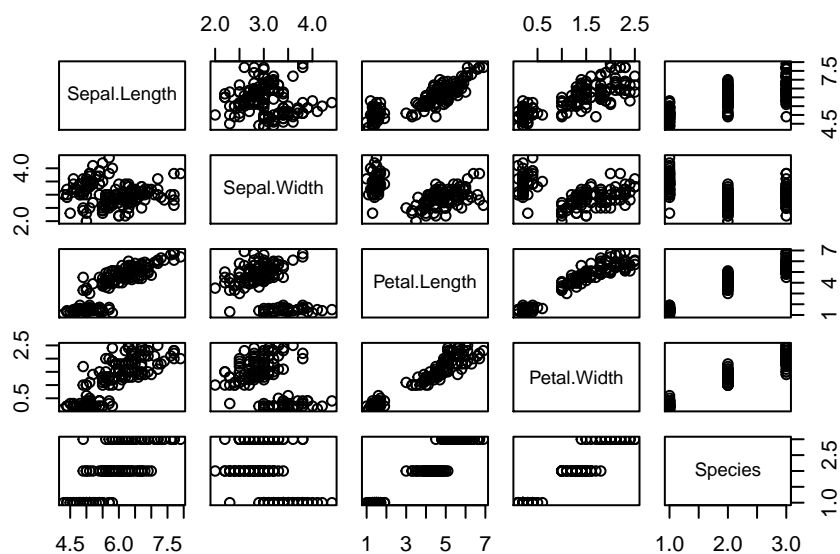
1. Base graphics
2. Lattice
3. ggplot2()

In this tutorial we will discuss only base graphics.

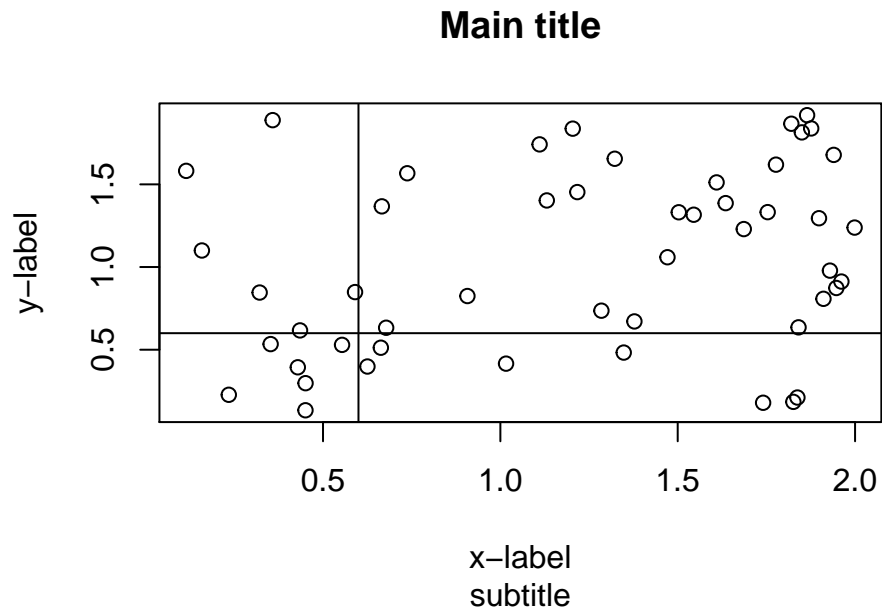
### 15.1 Plot

plot function is pretty intelligent. If you just give a dataframe, it will plot a graph with all vs all variables.

```
data(iris)
plot(iris)
```

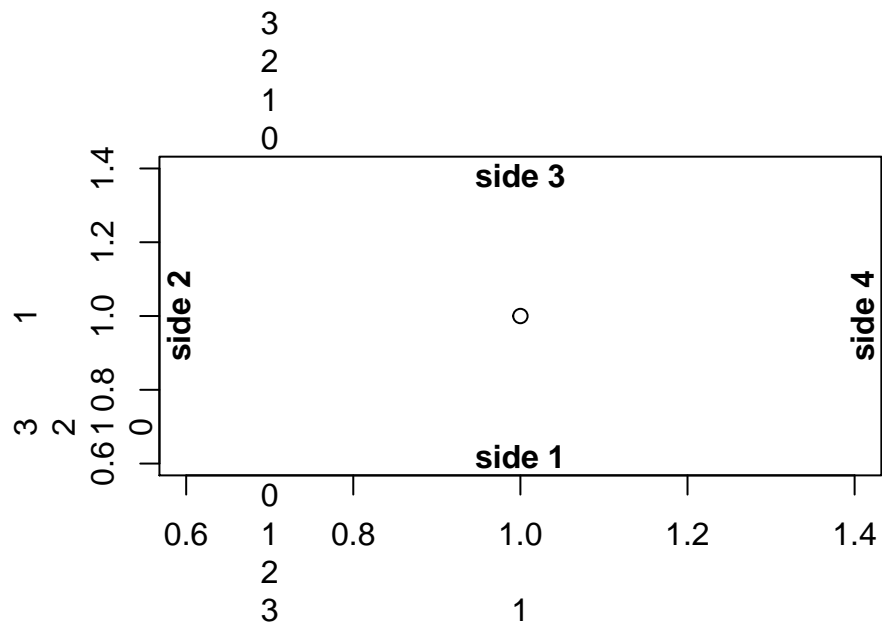


```
x<-runif(50,0,2) # create a vector from uniform distribution
y<-runif (50,0,2) # create another one
plot(x,y, main="Main title",sub="subtitle",xlab="x-label", ylab="y-label")
abline(h=0.6,v=0.6)
```



have specific margins.

```
plot(1,1)
for (side in 0:3) mtext (0:3,side=side,at=0.7,line=0:3)
mtext(paste("side",1:4),side=1:4,line = -1, font=2)
```

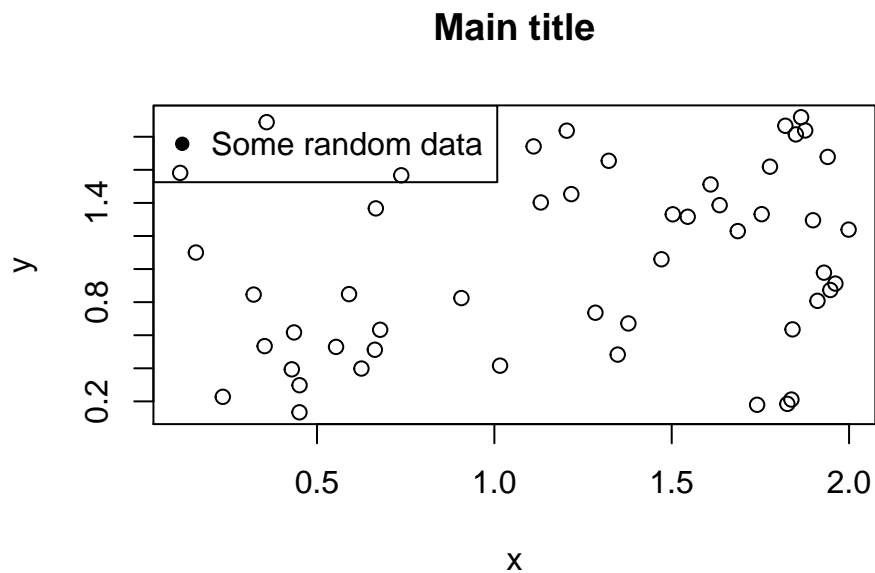


Each side of the plot



### 15.1.1 Building a plot from pieces

```
plot(x, y, type="n", xlab="", ylab="", axes=F) # Draw an empty plot area
points(x,y) # Draw point
axis(1) # Draw the first axis
axis(2, at=seq(0.2,1.8,0.2)) # Draw second axis with tick at specific scale
box() # Draw the surrounding box
title(main="Main title", xlab="x", ylab="y") # Write the main title and the labels
legend("topleft", legend="Some random data", pch=16)
```



### 15.1.2 Plotting characters

`pchShow()` will show you all the plotting characters.

## 15.2 Drawing devices in R

Normally when you create a plot, it is drawn on the default computer screen. But you can create all sorts of images, such as PDF (good for high resolution publication quality graphs), or bitmap images such as PNG. In all cases, the steps are identical. You open a device, draw your plot and close the device. For e.g., you can create a PDF drawing like this.

```
data(airquality)
pdf("airquality.pdf",width=8.5,height=11)
plot(airquality)
dev.off()
```

```
## pdf
## 2
```

### 15.3 par

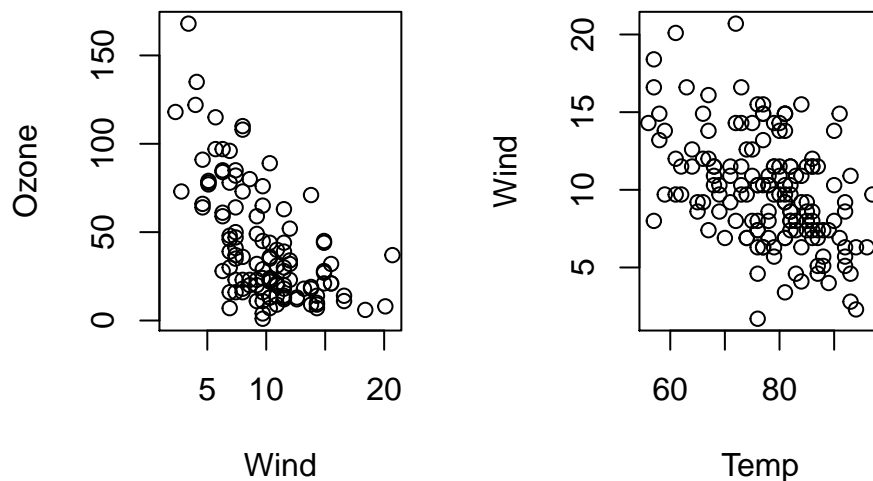
`par` function controls every parameter of a plot. The function has several parameters. But the most common ones are as follows:

1. `pch` - point type used for plotting
2. `lty` - Line type
3. `lwd` - line width
4. `cex` - character magnification. It has several variations for different regions of the plot.
5. `las` - rotation of the axis labels
6. `bg` - background color of the plot
7. `mar` - margin of the plot
8. `oma` - outer margin of the plot
9. `mfrow` - draws multiple plots

### 15.4 Drawing multiplots

You can modify `mfrow` in `par()`. For e.g, to draw a 1 row two column plot,

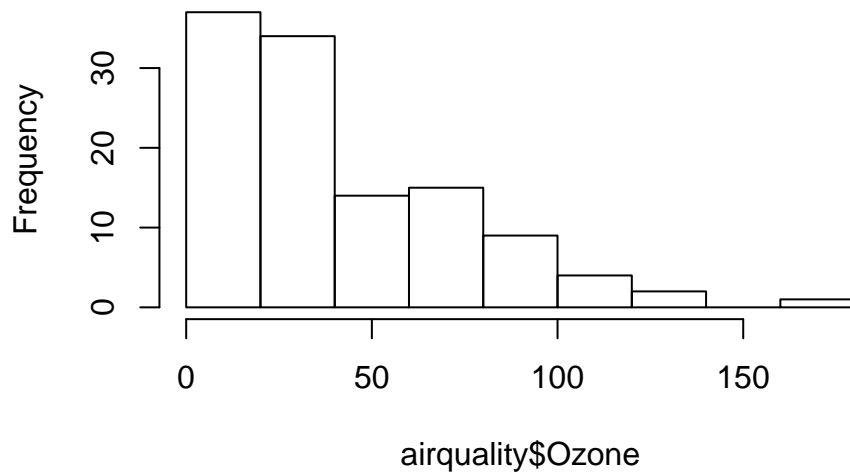
```
data(airquality)
par(mfrow=c(1,2))
with(airquality, {plot(Wind,Ozone);plot(Temp,Wind)})
```



### 15.5 Histograms

```
data("airquality")
hist(airquality$Ozone)
```

## Histogram of airquality\$Ozone

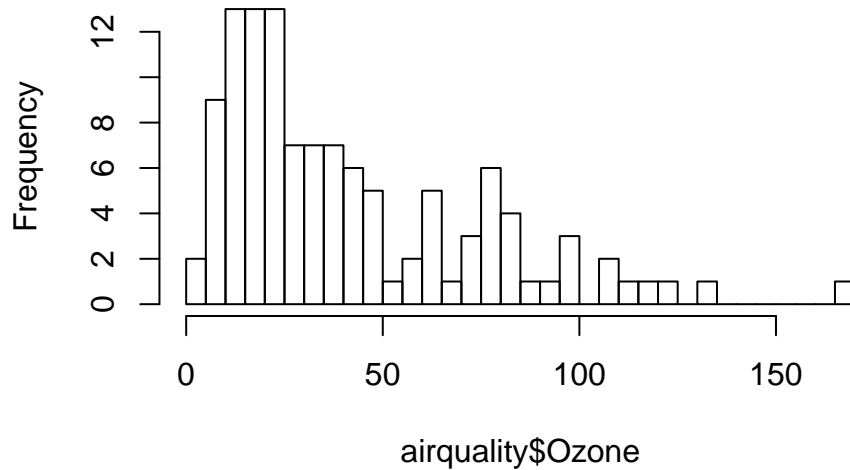


togram can be controlled:

```
hist(airquality$Ozone, breaks=50)
```

No of breaks in the his-

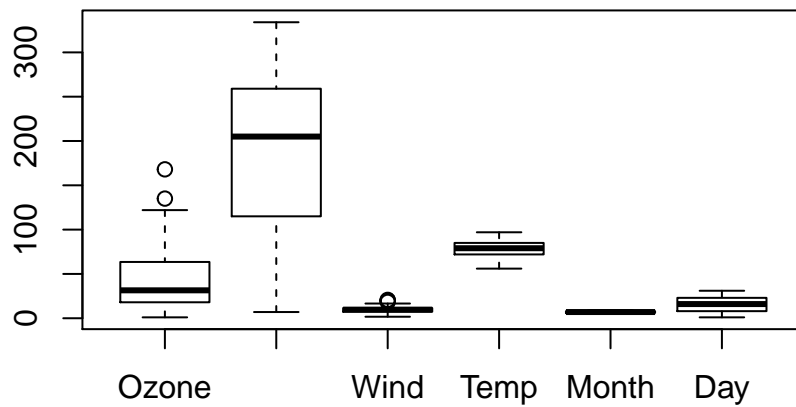
## Histogram of airquality\$Ozone



## 15.6 Boxplots

Boxplot is an summary plot showing the median and the distributions of data. You can give it a data frame.

```
boxplot(airquality)
```

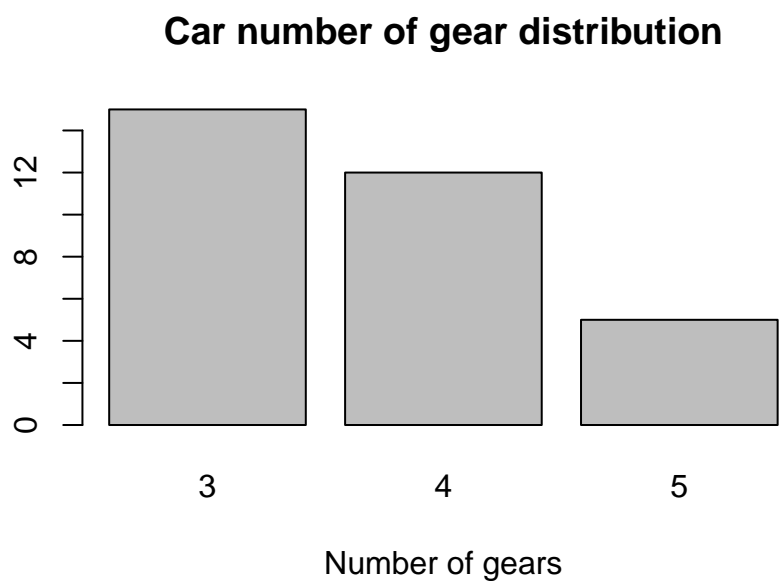


## 15.7 Barplots

```
data(mtcars)
head(mtcars)
```

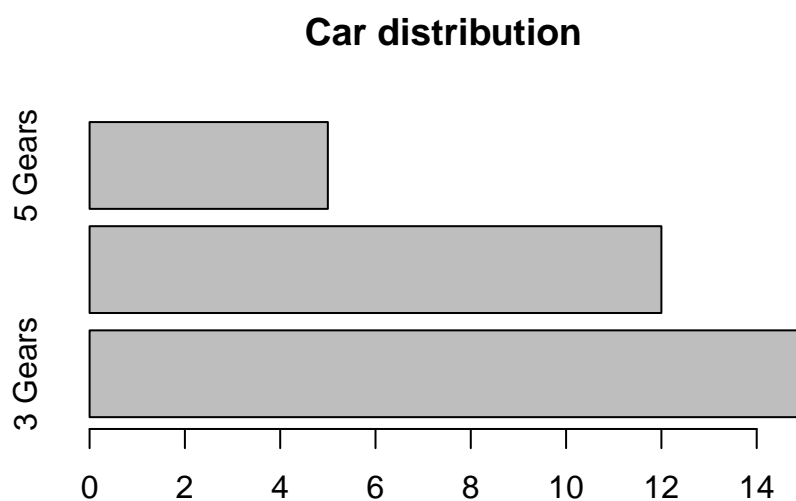
```
##           mpg cyl  disp  hp  drat   wt  qsec vs am gear carb
## Mazda RX4      21.0   6  160  110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6  160  110 3.90 2.875 17.02  0  1    4    4
## Datsun 710      22.8   4  108   93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive  21.4   6  258  110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360  175 3.15 3.440 17.02  0  0    3    2
## Valiant        18.1   6  225  105 2.76 3.460 20.22  1  0    3    1
```

```
counts<-table(mtcars$gear)
barplot(counts, main="Car number of gear distribution", xlab="Number of gears")
```



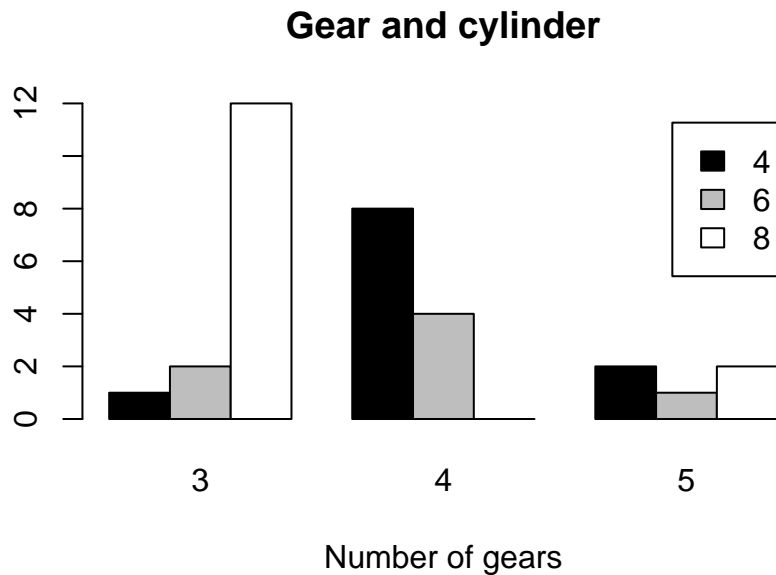
You can make a horizontal bar plot.

```
barplot(counts, main="Car distribution", horiz=T, names.arg=c("3 Gears", "4 Gears", "5 Gears"))
```



You can draw a grouped bar plot like this.

```
counts<-table(mtcars$cyl, mtcars$gear)
barplot(counts, main="Gear and cylinder",
        xlab="Number of gears",
        col=c("black","grey","white"),
        legend=rownames(counts),beside=T)
```



## 15.8 Barplots with error bars

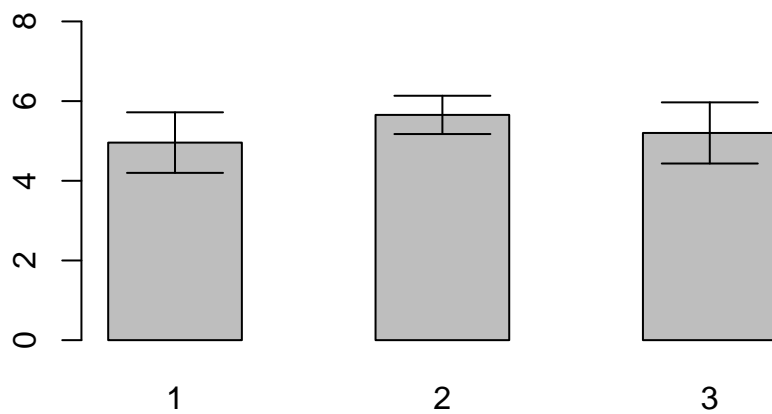
There is no inbuilt way to generate error bars in R. But we have to hack a solution. We will use arrows to generate error bars. The trick is arrows take an argument of `angle`. If you give `angle=90` it will generate an error bar.

```
## Generate some data
heights<-vector() # Empty vector to hold the heights of the bar plot
stddevs<-vector() # Empty vector to hold the standard deviations

for (i in 1:3) { # We will generate 3 sets of data
  d<-rnorm(10, mean=5, sd =1) # Be sure that we have some positive data
  heights[i]<-mean(d)
  stddevs[i]<-sd(d)
}

## Draw the barplot
bp<-barplot(heights,names=c(1:3),ylim=c(0,8),space=1) # Returns a matrix
mids<-bp[,1] # Get the midpoint

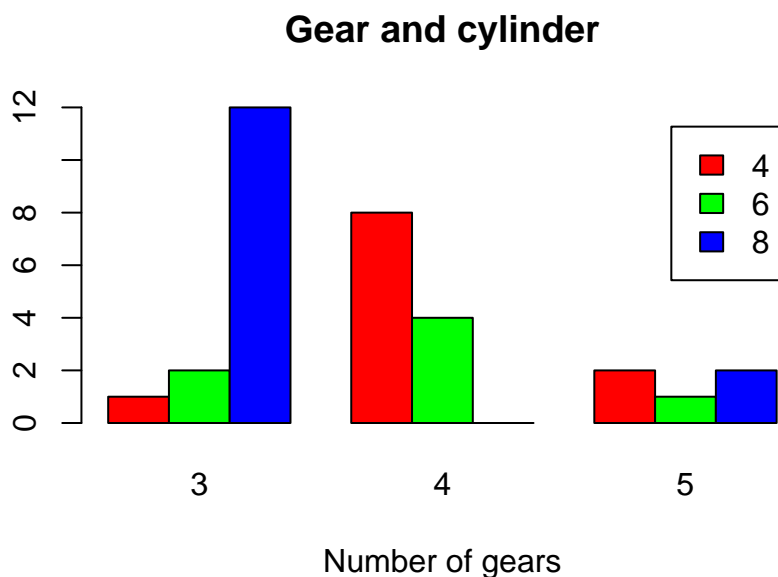
for (i in 1:3) { # We will draw the errorbars
  arrows(x0=mids[i],y0=heights[i]-stddevs[i],
        x1=mids[i],y1=heights[i]+stddevs[i],
        code=3,angle=90)
}
```



## 16 Colors in R

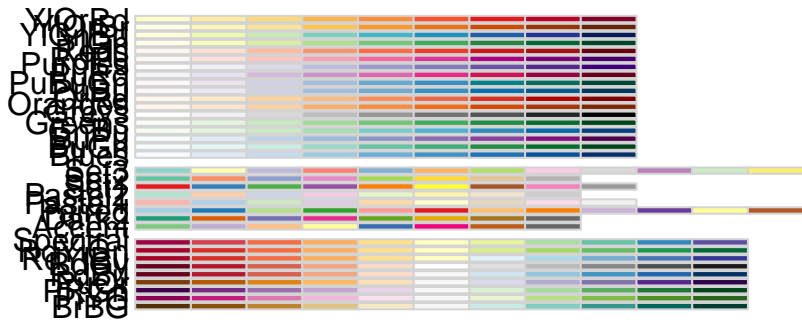
If you have drawn figures in Excel, you might have used colors like this

```
barplot(counts, main="Gear and cylinder", xlab="Number of gears", col=c("red","green","blue"),legend=ro
```



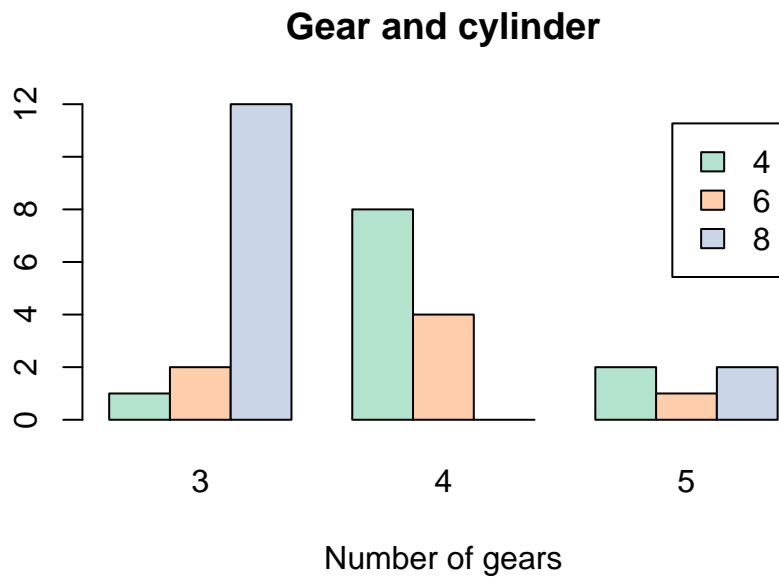
This is what I call “Christmas” plot. You should never use color like this. In R you can use sophisticated color palette.

```
library(RColorBrewer)
display.brewer.all()
```



There are 3 palettes: sequential, quantitative, and contrasting. Now choose a nice color for our plot.

```
nice<-brewer.pal(3,"Pastel2")
barplot(counts, main="Gear and cylinder", xlab="Number of gears", col=nice, legend=rownames(counts), beside=TRUE)
```





## 17 Linear regression and correlation

### 17.1 Linear regression

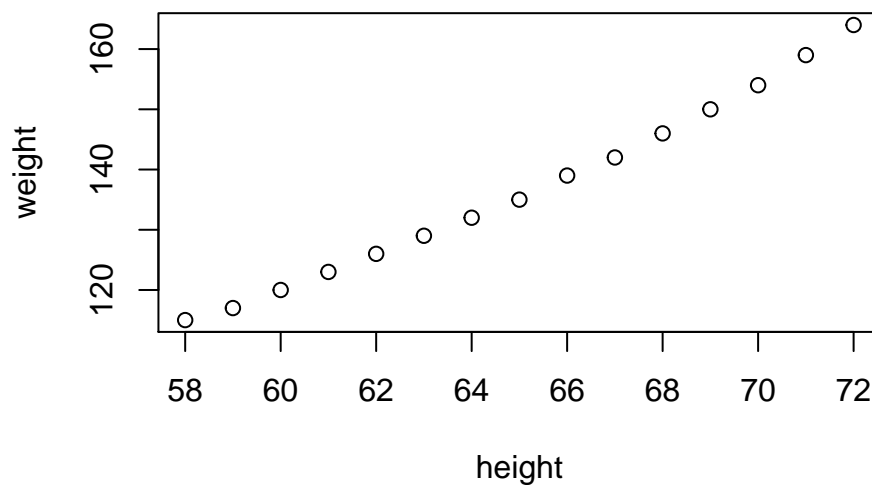
We will use the `women` datasets in R for this example. These is a simple datasets with “weight” and “height” of women in the US.

```
data(women)
head(women)
```

```
##   height weight
## 1     58    115
## 2     59    117
## 3     60    120
## 4     61    123
## 5     62    126
## 6     63    129
```

If we plot a simple scatterplot of this data, we will see there is an obvious correlation between this two variables.

```
plot(women)
```



sion between this two variables.

Let's do a linear regres-

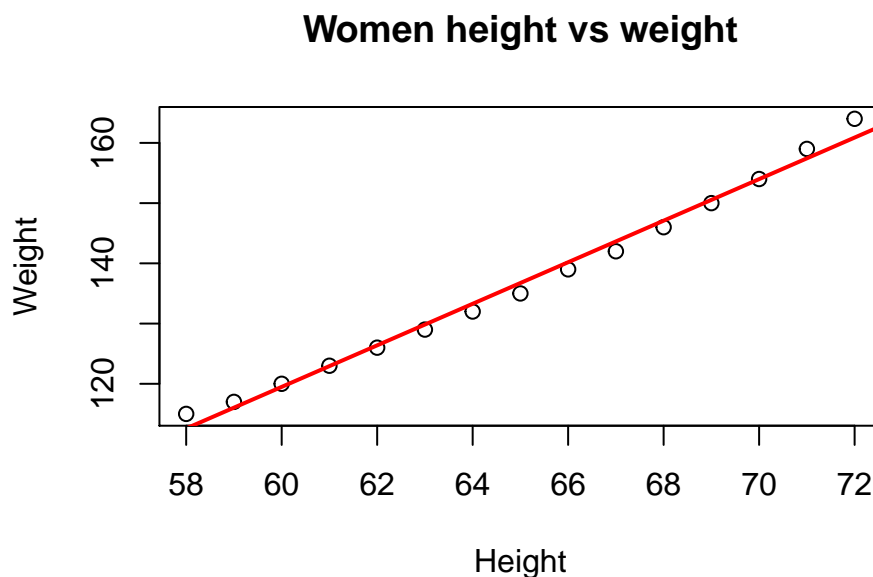
```
l <- lm(women$weight~women$height)
summary(l)
```

```
##
## Call:
## lm(formula = women$weight ~ women$height)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.7333 -1.1333 -0.3833  0.7417  3.1167
```

```
##
## Coefficients:
##           Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -87.51667     5.93694  -14.74 1.71e-09 ***
## women$height   3.45000     0.09114   37.85 1.09e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.525 on 13 degrees of freedom
## Multiple R-squared:  0.991, Adjusted R-squared:  0.9903
## F-statistic: 1433 on 1 and 13 DF, p-value: 1.091e-14
```

You can safely ignore the (Intercept) line. But the adjusted  $r^2$  is 0.99 with P-value  $1.1e-14$ , which is highly significant. Let's put the regression line on the plot.

```
plot(women$height,women$weight,main="Women height vs weight",xlab="Height",ylab="Weight")
abline(lwd=2,col="red")
```



You can extract the P-value and the  $r^2$  values for printing

```
s<-summary(l)
s$adj.r.squared

## [1] 0.9903183
s$coefficients[2,4]

## [1] 1.090973e-14
```

## 17.2 Correlations

We can also do a correlation between this two variables

```
s<-cor.test(women$weight,women$height,alternatives="two.sided")
s
```

```
##  
## Pearson's product-moment correlation  
##  
## data:  women$weight and women$height  
## t = 37.855, df = 13, p-value = 1.091e-14  
## alternative hypothesis: true correlation is not equal to 0  
## 95 percent confidence interval:  
##  0.9860970 0.9985447  
## sample estimates:  
##      cor  
## 0.9954948
```

```
s$p.value
```

```
## [1] 1.090973e-14
```

```
s$estimate
```

```
##      cor  
## 0.9954948
```

---