R Basics

Neba Nfonsang
University of Denver

In [1]:
```
require(dplyr, quiet=T)
install.packages("tidyverse", quiet=T)
require(tidyverse, quiet=T)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union


package 'tidyverse' successfully unpacked and MD5 sums checked

Loading tidyverse: ggplot2
Loading tidyverse: tibble
Loading tidyverse: tidyr
Loading tidyverse: readr
Loading tidyverse: purrr
Conflicts with tidy packages ----------------------------------------------------
filter(): dplyr, stats
lag():    dplyr, stats
```

R can be used as a calculator

In [2]:
```
# add
5 + 6
```

```
11
```

In [3]:
```
# subtract
5 - 3
```

```
2
```

In [4]:
```
# multiply
5 * 2
```

```
10
```

In [5]:
```
# divide
8/4
```

```
2
```

In [6]:
```
# modolus: returns a remainder
9%%2
```

```
1
```

In [7]:
```
# exponential
10^2
```

```
100
```

Using Built In Functions in R

In [8]:
```
# absolute value
abs(-5)
```

5

In [9]:
```
# natural log: log to the base e
log(10)
```

2.30258509299405

In [10]:
```
# log to a different base: log(num, base)
# log to the base 2
log(16, 2)
```

4

In [11]:
```
# log to the base 10
log10(10)
```

1

In [12]:
```
# square root
sqrt(25)
```

5

In [13]:
```
# e^x: e raise to the x
exp(2)
```

7.38905609893065

In [14]:
```
# factorial
factorial(5)
```

120

In [15]:
```
# round: to 2 decimal places
round(5.4532, 2)
```

5.45

In [16]:
```
# floor: takes only the whole part
floor(5.67)
```

5

In [17]:
```
# ceiling: rounds up
ceiling(5.67)
```

6

In [18]:
```
# trig function
cos(30)
sin(30)
tan(30)
```

0.154251449887584

-0.988031624092862

-6.40533119664628

In [19]:
```
# assignment
x <- 5
y <- 10
x
y
```

```
5
```

```
10
```

Missing Values

In [20]:
```
z <- c(1, 3, 5, 7, NA)
z
```

```
1  3  5  7  <NA>
```

In [21]:
```
# missing numbers affect the mean
mean(z)
```

```
<NA>
```

In [22]:
```
# remove the missing values while
# computing the mean
mean(z, na.rm=TRUE) # could use TRUE or T
```

```
4
```

Create a Vector

In [23]:
```
# using colon
a <- 1:10
a
```

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

In [24]:
```
# using concatenation function
b <- c(2, 4, 6, 8)
b
```

```
2  4  6  8
```

In [25]:
```
# name elements of a vector
names(b) <- c("A", "B", "C", "D")
b
```

```
A  2
B  4
C  6
D  8
```

In [26]:
```
# to remove names of elements in a vector
c <- as.vector(b)
c
```

```
2  4  6  8
```

Vector Functions in R

In [27]:
```
c
```

```
2  4  6  8
```

In [28]:
```
max(c)
min(c)
mean(c)
sum(c)
median(c)
range(c)
```

```
8
```

```
2
```

```
5
```

```
20
```

```
5
```

```
2  8
```

In [29]:
```
d <- c(20, 3, 5, 7, 1, 15)
```

In [30]:
```
sort(d)
```

```
1  3  5  7  15  20
```

In [31]:
```
# variance
var(d)
```

```
55.1
```

In [32]:
```
x <- 2:7
y <- c(20, 24, 26, 29, 30, 32)
```

In [33]:
```
# correlation
cor(x, y)
```

```
0.983837920767343
```

In [34]:
```
# load the dataset library
library(datasets)
```

In [35]:
```
# load a specific dataset: the car data
data(cars)
#view just the head of the data
head(cars)
```

speed	dist
4	2
4	10
7	4
7	22
8	16
9	10

Vector Functions for Data Frames

In [36]:
```
# column means
colMeans(cars)
```

speed	15.4
dist	42.98

In [37]:
```
# row means
rowMeans(cars)
```

3 7 5.5 14.5 12 9.5 14 18 22 14 19.5 13 16 18 20 19.5 23.5 23.5 29.5 20 25 37 47
17.5 20.5 34.5 24 28 24.5 28.5 33.5 30 37 47 51 27.5 32.5 43.5 26 34 36 38 42 44 38.5
47 58 58.5 72 55

In [38]:
```
# column totals
colSums(cars)
```

speed	770
dist	2149

In [39]:
```
# row totals
rowSums(cars)
```

6 14 11 29 24 19 28 36 44 28 39 26 32 36 40 39 47 47 59 40 50 74 94 35 41 69
48 56 49 57 67 60 74 94 102 55 65 87 52 68 72 76 84 88 77 94 116 117 144 110

In [40]:
```
# load the orange data
data(Orange)
head(Orange)
```

Tree	age	circumference
1	118	30
1	484	58
1	664	87
1	1004	115
1	1231	120
1	1372	142

In [41]:
```
# select circumference
cir <- Orange$circumference
cir
```

30 58 87 115 120 142 145 33 69 111 156 172 203 203 30 51 75 108 115 139 140 32
62 112 167 179 209 214 30 49 81 125 142 174 177

In [42]:
```
# count number of cases with circumference greater 150
sum(cir > 150)
```

10

In [43]:
```
# display circumferences greater than 150
cir[cir > 150]
```

156 172 203 203 167 179 209 214 174 177

In [44]:
```
# sum circumferences greater than 150
sum(cir[cir > 150])
```

1854

In [45]:
```
# find the indexes of cases with circumferences greater than 150
which(cir > 150)
```

```
11  12  13  14  25  26  27  28  34  35
```

In [46]:
```
# find the length of a vector
length(cir)
length(cir[cir > 150])
```

```
35
```

```
10
```

In [47]:
```
# indexes used to extract the values
cir[which(cir>150)]
# boolean values used to extract the values
cir[cir>150]
```

```
156  172  203  203  167  179  209  214  174  177
```

```
156  172  203  203  167  179  209  214  174  177
```

Repeats

In [48]:
```
# the function rep() is used to repeat value s
rep(20, 5)
```

```
20  20  20  20  20
```

In [49]:
```
# repeat 1 through 5, three times
rep(1:5, 3)
```

```
1  2  3  4  5  1  2  3  4  5  1  2  3  4  5
```

In [50]:
```
# repeat each number twice,
# for 3 iterations
rep(1:5, each=2, times=3)
```

```
1  1  2  2  3  3  4  4  5  5  1  1  2  2  3  3  4  4  5  5  1  1  2  2  3  3  4  4  5  5
```

Generate Letters

In [51]:
```
letters[1:26]
```

```
'a'  'b'  'c'  'd'  'e'  'f'  'g'  'h'  'i'  'j'  'k'  'l'  'm'  'n'  'o'  'p'  'q'  'r'  's'  't'  'u'  'v'  'w'  'x'  'y'  'z'
```

In [52]:
```
rep(letters[1:4], 5)
```

```
'a'  'b'  'c'  'd'  'a'  'b'  'c'  'd'  'a'  'b'  'c'  'd'  'a'  'b'  'c'  'd'  'a'  'b'  'c'  'd'
```

In [53]:
```
rep(letters[1:4], each=5)
```

```
'a'  'a'  'a'  'a'  'a'  'b'  'b'  'b'  'b'  'b'  'c'  'c'  'c'  'c'  'c'  'd'  'd'  'd'  'd'  'd'
```

Generate a Regular Sequence of Values

In [54]:
```
# use the seq() funtion
seq(from = 1, to = 4, by =0.2)
# or
seq(1, 4, 0.2)
```

```
1  1.2  1.4  1.6  1.8  2  2.2  2.4  2.6  2.8  3  3.2  3.4  3.6  3.8  4
```

```
1  1.2  1.4  1.6  1.8  2  2.2  2.4  2.6  2.8  3  3.2  3.4  3.6  3.8  4
```

In [55]:
```
# could also randomly generate a sequence of values
# rnorm(n, mean = 0, sd = 1)
rnorm(10, mean=100, sd=2)
```

```
102.657951954303  102.5907563933  98.5470883118932  100.474263469201  98.0672044112067
99.3260716047889  98.8758391383971  98.3329108120193  99.5541836794872  98.3016937474658
```

In [56]:
```
# generate values from a standard normal distribution
# with mean of 0 and standard deviation of 1
rnorm(10)
```

```
-1.07281539487197  0.265658446467568  -0.409344228738537  -1.38192179757164  -0.53351559225933
-0.788203992601643  -0.0477629708004704  -1.0382822420375  -0.00375393742709201  -0.145421071175307
```

Matrices

The matrix() function is used to create matrices. Basically, the matrix() function takes a vector, number of rows and number of columns as arguments.

In [57]:
```
# matrix(data=NA, nrow=1, ncol=1, byrow=FALSE,dimnames = NULL)
mat <- matrix(1:10, nrow=2)
mat
```

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

In [58]:
```
mat <- matrix(1:12, ncol=3)
mat
```

```
1  5  9
2  6  10
3  7  11
4  8  12
```

In [59]:
```
# we could include both columns and rows
# just to be explict but specifying just the row or column
# produces the same results
mat <- matrix(1:12, nrow= 4, ncol=3)
mat
```

```
1  5  9
2  6  10
3  7  11
4  8  12
```

In [60]:
```
# byrow determines how the values should
# be entered
mat <- matrix(1:12, nrow= 4, byrow=T)
mat
```

1	2	3
4	5	6
7	8	9
10	11	12

In [61]:
```
row.names <- letters[1:4]
col.names <- c("x", "y", "z")
mat <- matrix(1:12, nrow= 4, byrow=T,
              dimnames = list(row.names, col.names) )
mat
```

	x	y	z
a	1	2	3
b	4	5	6
c	7	8	9
d	10	11	12

In [ ]:
```

```

Character String

In [ ]:
```

```

for loops

In [62]:
```
# for (loopvar in iter){
        #do something
#}

for (i in 1:4){
    squares = i^2
    print(squares)
}
```

```
[1] 1
[1] 4
[1] 9
[1] 16
```

In [63]:
```
# initalize an empty vector
square.vector <- rep("NA", 4)
for (i in 1:4){
    square.vector[i] <- i^2
}
square.vector
```

'1' '4' '9' '16'

while loops

In [64]:
```
# create a loops that add up numbers
# from 1 to 4
x = 0
tot = 0
while (x<5){
    tot = tot + x
    x = x + 1
}
tot
```

10

Conditional if/else Statements

In [65]:
```
x = -5
if (x>0){
    sprintf("%s is a positive number", x)
} else if (x<0){
    sprintf("%s is a negative number", x)
} else{
    sprintf("%s is zero", x)
}
```

'-5 is a negative number'

Writing Functions in R

In [66]:
```
# function(param1, param2,...){
    # do something
#}

average <- function(a, b){
    tot = (a + b)/2
    tot
}

average(4, 5)
```

4.5

Data Frames

In [67]:
```
w <- 1:6
x <- 21:26
y <- 31:36
z <- rep(c("A", "B"), each=3)
dat <- data.frame(w, x, y, z)
head(dat)
```

	w	x	y	z
1	1	21	31	A
2	2	22	32	A
3	3	23	33	A
4	4	24	34	B
5	5	25	35	B
6	6	26	36	B

In [68]:
```
# slice the frame
dat$w
dat$x
dat$y
```

```
1  2  3  4  5  6
```

```
21  22  23  24  25  26
```

```
31  32  33  34  35  36
```

Slicing a Data Frame

- dat[column position] gets a column
- dat[column name] gets a column
- dat[c(colname1, colname2, etc] gets multiple columns
- dat[row position, col poisition] get data from a specific position
- dat[row position,] gets data from a row
- dat[, col position] gets data for a column
- dat[, c(row1, row2 etc] gets data from multiple columns

In [69]:
```
# get the first column as dataframe
dat[1]
```

w
1
2
3
4
5
6

In [70]:
```
# get the first column as a vector dat[row, col]
# what comes before comma is rows,
# then after the comma is columns
dat[,1]
```

```
1  2  3  4  5  6
```

In [71]:
```
# second row, first column
dat[2, 1]
```

```
2
```

In [72]:
```
# get first and third column as data frame
dat[c(1, 3)]
```

w	y
1	31
2	32
3	33
4	34
5	35
6	36

In [73]:
```
# get first and third rows
# having no value after the comma indicates all columns
dat[c(1, 3),]
```

	w	x	y	z
1	1	21	31	A
3	3	23	33	A

In [74]:
```
# values from row1 to row 4 and
# from column 1 to column 2
dat[1:4, 1:2]
```

w	x
1	21
2	22
3	23
4	24

In [75]:
```
# can also get columns using column names
# inside the square brackets
dat[c("x", "y")]
```

x	y
21	31
22	32
23	33
24	34
25	35
26	36

In [76]:
```
# get all columns except column 1
dat[-1]
```

x	y	z
21	31	A
22	32	A
23	33	A
24	34	B
25	35	B
26	36	B

In [77]:
```
# select certain rows based on a boolean expression
# returns the rows where x is greater than 23
dat[dat$x>23,]
```

	w	x	y	z
4	4	24	34	B
5	5	25	35	B
6	6	26	36	B

In [78]:
```
# use the select function to select columns
# select is from the dplyr package
select(dat, x, z)
```

x	z
21	A
22	A
23	A
24	B
25	B
26	B

In [79]:
```
# use the which() function to select rows
# that meet a certain boolean condition
which(dat$x>=24) # returns row index
```

4 5 6

In [80]:
```
dat[which(dat$x>=24),]
```

	w	x	y	z
4	4	24	34	B
5	5	25	35	B
6	6	26	36	B

In [81]:
```
dat[which(dat$z=="A"),]
```

w	x	y	z
1	21	31	A
2	22	32	A
3	23	33	A

Sorting

In [82]:
```
library(datasets)
data(cars)
head(cars)
```

speed	dist
4	2
4	10
7	4
7	22
8	16
9	10

In [83]:
```
# sort the second column
sort(cars$dist)
```

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

In [84]:
```
# gives the index of the data in sorted order
ind <- order(cars$dist)
ind
```

```
 1  3  2  6 12  5 10  7 13 24  4 14  8 16 20 25 11 15 27 29 39  9 17 18 21 36 28 30
32 19 37 40 31 41 26 45 33 42 22 43 44 38 46 34 23 35 50 47 48 49
```

In [85]:
```
# sort the data by distance
# use the sorted indexes of distance
head(cars[ind,])
```

	speed	dist
1	4	2
3	7	4
2	4	10
6	9	10
12	12	14
5	8	16

Use the arrange function to sort

In [86]:
```
require(dplyr, quiet=T)
install.packages("tidyverse", quiet=T)
require(tidyverse, quiet=T)
```

```
Warning message:
"package 'tidyverse' is in use and will not be installed"
```

In [87]:
```
# this orders by speed, and within each speed,
# the data is ordered by area
ordered.speed.dist <- arrange(cars, speed, dist)
head(ordered.speed.dist)
```

speed	dist
4	2
4	10
7	4
7	22
8	16
9	10

List all objects or variables created

In [88]:
```
ls()
```

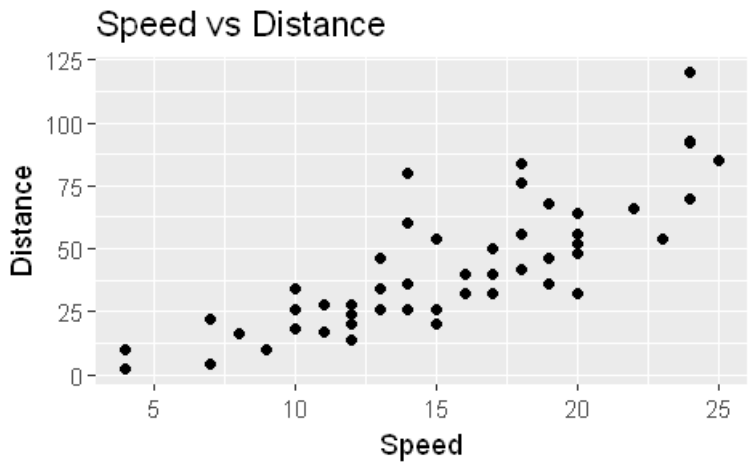
```
'a' 'average' 'b' 'c' 'cars' 'cir' 'col.names' 'd' 'dat' 'i' 'ind' 'mat' 'Orange' 'ordered.speed.dist' 'row.names'
'square.vector' 'squares' 'tot' 'w' 'x' 'y' 'z'
```

In [ ]:
```

```

Use quick plots

In [89]:
```
# save plot as png
# png(file="myplot.png", bg="transparent")
# plot settings
options(repr.plot.width=4, repr.plot.height=2.5)
# quick plot
qplot(cars$speed, cars$dist,
      xlab="Speed", ylab="Distance", main="Speed vs Distance")
dev.off()
```

null device: 1



The Apply Function

Used to apply a function across rows or columns. It works with dataframes or matrices

In [90]:
```
# draw numbers from a normal distribution
set.seed(2020)
values <- rnorm(n=20, mean=10, sd=4)
# create a matrix with the values
mat <- matrix(values, ncol=5)
mat
```

11.507888	-1.186137	17.036525	14.785492	16.8159835
11.206193	12.882294	10.469467	8.513664	-2.1550584
5.607907	13.756484	6.587509	9.506959	0.8441002
5.478376	9.082489	13.637037	17.200172	10.2332140

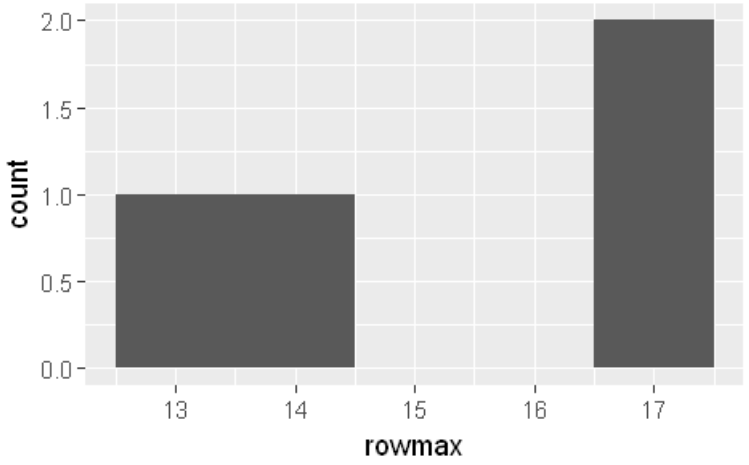
In [91]:
```
# margin=1 if you are applying the function by row
# margin=2 if you are applying the function by col

# find row maximum values
apply(mat, MARGIN = 1, FUN = max)

# find row mean
apply(mat, MARGIN = 1, FUN = mean)

# find col mean
apply(mat, MARGIN = 2, FUN = mean)
```

17.0365253878539 12.8822939936463 13.756484092036 17.2001724669018

11.7919503911002 8.18331212008893 7.26059187748974 11.1262577144203

8.45009142461471 8.63378245484136 11.9326344930975 12.5015719392494 6.43455981707086

In [92]:
```
# plot the count rowmax bar chart
rowmax <- apply(mat, MARGIN = 1, FUN = max)
qplot(rowmax, binwidth=1)
```



The lapply() and the sapply() functions

lapply() applies a function to a list, and outputs a list. sapply() is similar to lapply but outputs a vector.

In [93]:
```
# create a list
mylist <- list("A", TRUE, FALSE, 9, 8.7)

# find the class of each value in the list
lapply(mylist, class)
```

1. 'character'
2. 'logical'
3. 'logical'
4. 'numeric'
5. 'numeric'

In [94]:
```
sapply(mylist, class)
```

'character' 'logical' 'logical' 'numeric' 'numeric'

In [95]:
```
my.numbers <- 1:5
sapply(my.numbers, sqrt)
```

1 1.4142135623731 1.73205080756888 2 2.23606797749979

In [96]:
```
# define a function to return the
# a number raised to the 5th power
func <- function(num){
    num^5
}
# apply the function to numbers
sapply(my.numbers, func)
```

```
1  32  243  1024  3125
```

In [97]:
```
# lets apply this to the data frame
dat.numeric <- select(dat, w, x, y)
head(dat.numeric)
```

w	x	y
1	21	31
2	22	32
3	23	33
4	24	34
5	25	35
6	26	36

In [98]:
```
# raise each value to the 5th power
sapply(dat.numeric, func)
```

w	x	y
1	4084101	28629151
32	5153632	33554432
243	6436343	39135393
1024	7962624	45435424
3125	9765625	52521875
7776	11881376	60466176

In [99]:
```
apply(dat.numeric, 2, func)
```

w	x	y
1	4084101	28629151
32	5153632	33554432
243	6436343	39135393
1024	7962624	45435424
3125	9765625	52521875
7776	11881376	60466176

In [100]:
```
lapply(dat.numeric, func)
```

```
$w
1  32  243  1024  3125  7776

$x
4084101  5153632  6436343  7962624  9765625  11881376

$y
28629151  33554432  39135393  45435424  52521875  60466176
```

Note that lapply() and sapply() applies a function to each element of a sequence but apply() applies a function to an entire row or column vector so aggregrate functions such as mean() can be used with the apply() function, but not with the lapply() and sapply().

In [101]:
```
# return the second element of every column
sapply(dat, function(col){col[2]})
```

w	2
x	22
y	32
z	1

In [102]:
```
# apply mean to each row, use apply() for row or column operations
apply(dat.numeric, 2, mean)
```

w	3.5
x	23.5
y	33.5

The summarize() function

This function is in dplr and works works with a group_by parameter to group data by a categorical variable then find aggregates such as mean within each group.

In [103]:
```
# view data again
head(dat)
```

w	x	y	z
1	21	31	A
2	22	32	A
3	23	33	A
4	24	34	B
5	25	35	B
6	26	36	B

In [104]:
```
# summarize the data: first group by values of z,
# then within each group, find the mean of x
summarize(group_by(dat, z), mean=mean(x))
```

z	mean
A	22
B	25

In [105]:
```
summarize(dat)
```

The filter() function

Works like a boolean selection

In [106]:
```
filter(dat, z=="A")
```

w	x	y	z
1	21	31	A
2	22	32	A
3	23	33	A

The attach() function

- Using the attach() function on the data frame helps you access the columns by just using the column names without attaching them to the name of the data frame. Note that the detach() function does the opposite of what the attach()

function does. The detach() function removes the names of the column variables of a data frame from the name space.

In [107]:
```
attach(cars)
```

In [108]:
```
# access the values in the speed column
cars$speed
```

```
 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 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
```

In [109]:
```
# access the values in the speed column
speed
```

```
 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 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
```

Read data into R

Read and write csv files

- read_csv(file)

Read and write sas files

- read_sas(data_file, catalog_file = NULL, encoding = NULL)
- write_sas(data, path)

Read and write spss files

- read_sav(file, user_na = FALSE)
- read_por(file, user_na = FALSE)
- read_spss(file, user_na = FALSE)
- write_sav(data, path)

Save Data in to CSV

In [110]:
```
path = "C:/Users/nnfon/Desktop/R_Programming/cars_data.csv"
write_csv(cars, path)
```

Data Types

- use the class() function to view the data type of an object
- use the str() to view the structure of an object

In [111]:
```
class(dat)
class(mat)
class(dat$x)
class(dat$z)
```

'data.frame'

'matrix'

'integer'

'factor'

In [112]:
```
# view the iris dataset
head(iris)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

In [113]:
```
str(iris)
```

```
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

In [114]:
```
# check the levels or categorical values
# in the species column
levels(iris$Species)
```

```
'setosa'   'versicolor'   'virginica'
```

In [115]:
```
# check the dimensions of the data
dim(iris)
```

```
150   5
```

In [ ]: