ml-class-02.Rmd

Mariano Dominguez

August 19, 2015

R Statistics

A basic notion then is that of a random sample.

In R, you can simulate these situations with the sample function. If you want to pick five numbers at random from the set 1:40, then you can write:

```
sample(1:40,5)
```

```
## [1]  3 18 17  9 10
```

Sampling with replacement is suitable for modelling coin tosses or throws of a die. So, for instance, to simulate 10 coin tosses we could write

```
sample(c("H","T"), 10, replace=T)
```

```
##  [1] "H" "H" "T" "T" "T" "H" "T" "H" "T" "H"
```

You can simulate data with nonequal probabilities for the outcomes (say, a 90% chance of success) by using the prob argument to sample, as in

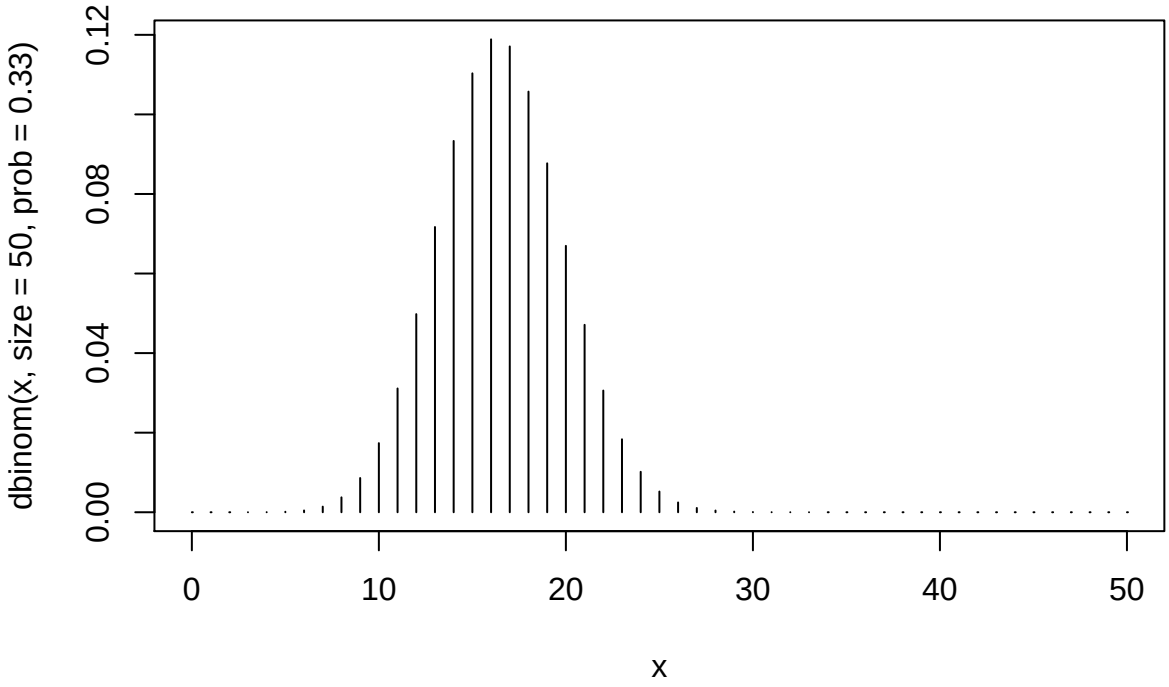
```
sample(c("succ", "fail"), 10, replace=T, prob=c(0.9, 0.1))
```

```
##  [1] "succ" "succ" "succ" "succ" "fail" "fail" "succ" "succ" "succ" "succ"
```

R Discrete distributions

where variables can take on only distinct values, it is preferable to draw a pin diagram, here for the binomial distribution with n = 50 and p = 0.33 (Figure 3.2):

```
x <- 0:50
plot(x,dbinom(x,size=50,prob=.33),type="h")
```



Some Random numbers

To many people, it sounds like a contradiction in terms to generate random numbers on a computer since its results are supposed to be predictable and reproducible. What is in fact possible is to generate sequences of "pseudo-random" numbers, which for practical purposes behave as if they were drawn randomly.

In statistics, they are used to create simulated datasets in order to study the accuracy of mathematical approximations and the effect of assumptions being violated. The use of the functions that generate random numbers is straightforward. For example, the normal variate generation function is:

```
n=100
x <- rnorm (n, mean=0, sd=1)
mean(x)
```

```
## [1] 0.03332448
```

```
sd(x)
```

```
## [1] 1.0829
```

```
quantile(x)
```

```
##          0%         25%         50%         75%        100%
## -2.5698430 -0.6788525  0.1235249  0.7742423  2.6923594
```

Subsetting

The subset(), which() and ifelse() are probably the most commonly used functions in R. One way to filter elements in the vector is to use subset() function.

```
# create a vector
x <- c(5,4:8,12)
x
```

```
## [1]  5  4  5  6  7  8 12
```

```
y <- subset(x, x < 6)
y
```

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

By using which() we identify the position in vector if the condition is TRUE. Follow the example below to see how to use it.

```
# create a vector
z <- c(6:10, 12, -3)
z
```

```
## [1]  6  7  8  9 10 12 -3
```

```
which(z > 8)
```

```
## [1] 4 5 6
```

The ifelse has two statements to execute. If condition is TRUE the first statement is executed. If condition is FALSE, the second statement is executed. The syntax is ifelse(condition, statement1, statement2). An example is below.

```
# create a vector
x <- c(-2, 5:10, 15)
x
```

```
## [1] -2  5  6  7  8  9 10 15
```

```r
# if values are < 7 will code those 1, else will become 0
ifelse(x < 7, 1, 0)
```

```
## [1] 1 1 1 0 0 0 0 0
```

```r
# also you can do this
ifelse(x < 7, 1, x)
```

```
## [1]  1  1  1  7  8  9 10 15
```

Code the Matrix

To assess row and columns in matrices use this function x[row,column]. In example below I will create a matrix by using the function rnorm() which is a function to create random numbers from normal distribution.

```r
# matrix with 12 random numbers in 4 rows
x <- matrix(rnorm(12), nrow=4)
x
```

```
##            [,1]       [,2]       [,3]
## [1,]  0.3616641  0.5826593  1.7673954
## [2,] -0.9815806 -0.2857379 -0.4653591
## [3,]  0.6336556 -0.9633035 -0.8826791
## [4,] -0.8897125  0.5690305  0.3144519
```

```r
# find the number in 3rd row and 2nd column
x[3,2]
```

```
## [1] -0.9633035
```

Also, you can show entire column or row. Using the matrix from example above we now identify entire column or row.

```r
# show second columns
x[,2]
```

```
## [1]  0.5826593 -0.2857379 -0.9633035  0.5690305
```

```r
# show forth row
x[4,]
```

```
## [1] -0.8897125  0.5690305  0.3144519
```

Finally I would like to show another interesting function dim() which is used to identify the number of rows and columns in matrices.

```r
# matrix from example above
x
```

```
##            [,1]       [,2]       [,3]
## [1,]  0.3616641  0.5826593  1.7673954
## [2,] -0.9815806 -0.2857379 -0.4653591
## [3,]  0.6336556 -0.9633035 -0.8826791
## [4,] -0.8897125  0.5690305  0.3144519
```

```r
# find number or columns and rows in matrix
dim(x)
```

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

Later we will come back to matrix operations.

Data Import and Export

Data in R can be saved as .Rdata files with functions save. After that, they can then be loaded into R with load.

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

The example below creates a dataframe a and save it as a .CSV file with write.csv. And then, the dataframe is loaded from file to variable b with read.csv.

```r
var1 <- 1:5
var2 <- (1:5) / 10
var3 <- c("R", "and", "Data Mining", "Examples", "Case Studies")
a <- data.frame(var1, var2, var3)
names(a) <- c("VariableInt", "VariableReal", "VariableChar")
write.csv(a, "Data.csv", row.names = FALSE)
 #rm(a)
b <- read.csv("Data.csv")
print(b)
```

```
##   VariableInt VariableReal VariableChar
## 1           1          0.1            R
## 2           2          0.2          and
## 3           3          0.3  Data Mining
## 4           4          0.4     Examples
## 5           5          0.5 Case Studies
```

Create un Data Frame

Data frame are similar to matrix, but in comparison with matrix, data frame contain numeric and character elements. Therefore, a data frame can have one column with numbers and other column with a characters. The function used to create data frames is dataframe() Let's create a simple data frame.

```
# create a data frame
hospital <- c("New York", "California")
patients <- c(150, 350)
df <- data.frame(hospital, patients)
df
```

```
##     hospital patients
## 1   New York      150
## 2 California      350
```

```
# structure
str(df)
```

```
## 'data.frame':    2 obs. of  2 variables:
##  $ hospital: chr  "New York" "California"
##  $ patients: num  150 350
```

Read Write Table

La funcion write.table guarda el contenido de un objeto en un archivo. El objeto es tıpicamente un marco de datos ('data.frame'), pero puede ser cualquier otro tipo de objeto (vector, matriz,. . .).

```
write.table(df,file="data.dat")
```

La funcion read.table crea un marco de datos ('data frame') y constituye la manera mas usual de leer datos en forma tabular. Por ejemplo si tenemos un archivo de nombre data.dat, el comando: (tome un archivo de datos y copielo a ./data.dat)

```
misdatos <- read.table("data.dat")
misdatos$hospital
```

```
## [1] "New York"   "California"
```

```
misdatos["hospital"]
```

```
##     hospital
## 1   New York
## 2 California
```

creara un marco de datos denominado misdatos, y cada variable recibira por defecto el nombre V1, V2, . . . y puede ser accedida individualmente escribiendo misdatos$V1, misdatos$V2, . . . , o escribiendo misdatos["V1"], misdatos["V2"], . . . , o, tambien escribiendo misdatos[,1], misdatos[,2], .. etc

Existen varias opciones con valores por defecto (aquellos usados por R si son omitidos por el usuario). Look for Help!

```
?read.table
```

Some intrinsic Datasets

iris dataset

```
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 ...
```

bodyfat dataset, brief excursion to install.packages("") details:

```
#library(mboost)
data("bodyfat", package="TH.data")
str(bodyfat)
```

```
## 'data.frame':    71 obs. of  10 variables:
##  $ age         : num  57 65 59 58 60 61 56 60 58 62 ...
##  $ DEXfat      : num  41.7 43.3 35.4 22.8 36.4 ...
##  $ waistcirc   : num  100 99.5 96 72 89.5 83.5 81 89 80 79 ...
##  $ hipcirc     : num  112 116.5 108.5 96.5 100.5 ...
##  $ elbowbreadth: num  7.1 6.5 6.2 6.1 7.1 6.5 6.9 6.2 6.4 7 ...
##  $ kneebreadth : num  9.4 8.9 8.9 9.2 10 8.8 8.9 8.5 8.8 8.8 ...
##  $ anthro3a    : num  4.42 4.63 4.12 4.03 4.24 3.55 4.14 4.04 3.91 3.66 ...
##  $ anthro3b    : num  4.95 5.01 4.74 4.48 4.68 4.06 4.52 4.7 4.32 4.21 ...
##  $ anthro3c    : num  4.5 4.48 4.6 3.91 4.15 3.64 4.31 4.47 3.47 3.6 ...
##  $ anthro4     : num  6.13 6.37 5.82 5.66 5.91 5.14 5.69 5.7 5.49 5.25 ...
```

```
head(bodyfat)
```

```
##    age DEXfat waistcirc hipcirc elbowbreadth kneebreadth anthro3a anthro3b
## 47  57  41.68     100.0   112.0          7.1         9.4     4.42     4.95
## 48  65  43.29      99.5   116.5          6.5         8.9     4.63     5.01
## 49  59  35.41      96.0   108.5          6.2         8.9     4.12     4.74
## 50  58  22.79      72.0    96.5          6.1         9.2     4.03     4.48
## 51  60  36.42      89.5   100.5          7.1        10.0     4.24     4.68
## 52  61  24.13      83.5    97.0          6.5         8.8     3.55     4.06
##    anthro3c anthro4
## 47     4.50    6.13
## 48     4.48    6.37
## 49     4.60    5.82
## 50     3.91    5.66
## 51     4.15    5.91
## 52     3.64    5.14
```

Data exploration

Have a Look at Data * 1-Check the dimensionality

```
dim(iris)
```

```
## [1] 150   5
```

- 2 Variable names or column names

```
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

- 3 Structure

```
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 ...
```

- 4 Attributes

```
attributes(iris)
```

```
## $names
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
##
## $row.names
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102
## [103] 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
## [120] 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136
## [137] 137 138 139 140 141 142 143 144 145 146 147 148 149 150
##
## $class
## [1] "data.frame"
```

- 5 Get the first 5 rows

```r
iris[1:5,]
```

```
##   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 Get Sepal.Length of the first 10 rows

```r
iris[1:10, "Sepal.Length"]
```

```
##  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
```

Explore Individual Variables

- 1 Distribution of every variable

```r
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

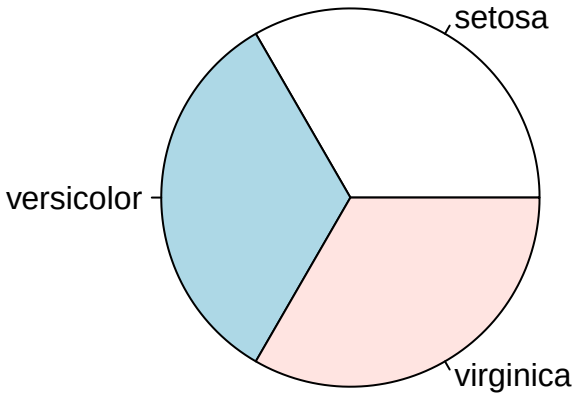
- 2 Frequency

```r
table(iris$Species)
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

- 3 Pie chart

```
pie(table(iris$Species))
```



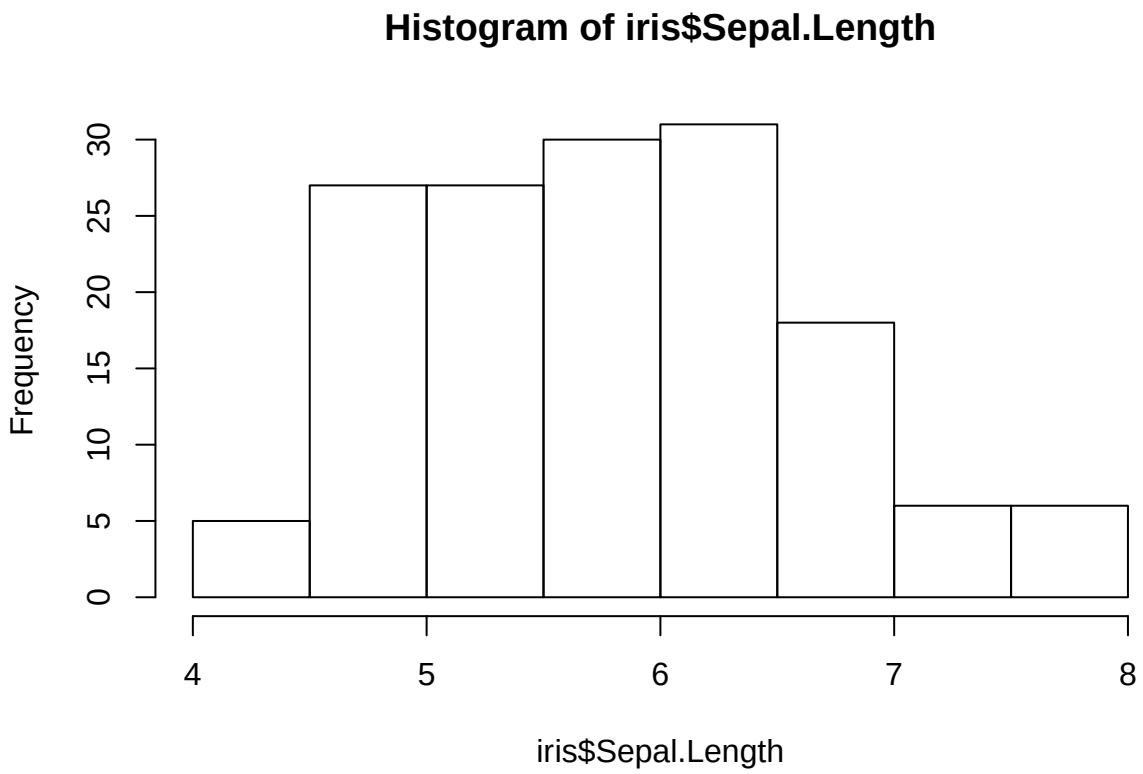
* 4 Variance of Sepal.Length

```
var(iris$Sepal.Length)
```

```
## [1] 0.6856935
```

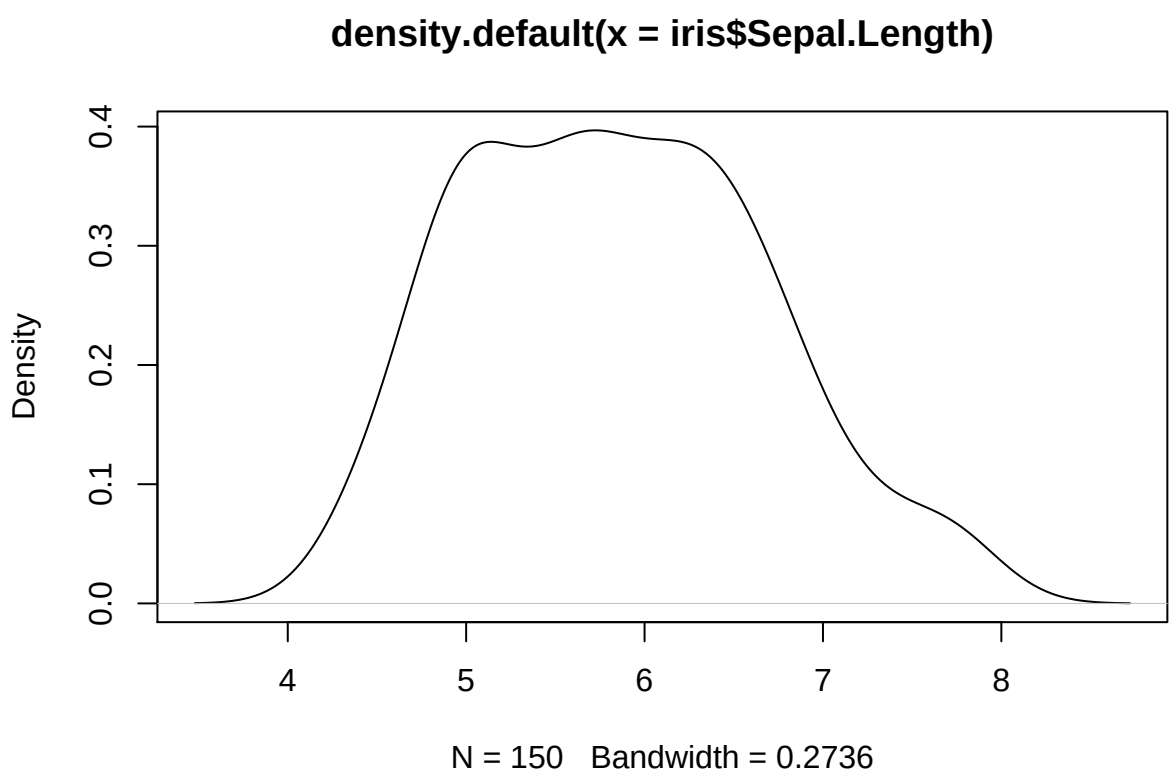
- 5 Histogram

```
hist(iris$Sepal.Length)
```



* 6 Density

```
plot(density(iris$Sepal.Length))
```



** Explore Multiple Variables

- 1 covariance of two variables

```
cov(iris$Sepal.Length, iris$Petal.Length)
```

```
## [1] 1.274315
```

- 2 Correlation of two variables

```
cor(iris$Sepal.Length, iris$Petal.Length)
```

```
## [1] 0.8717538
```

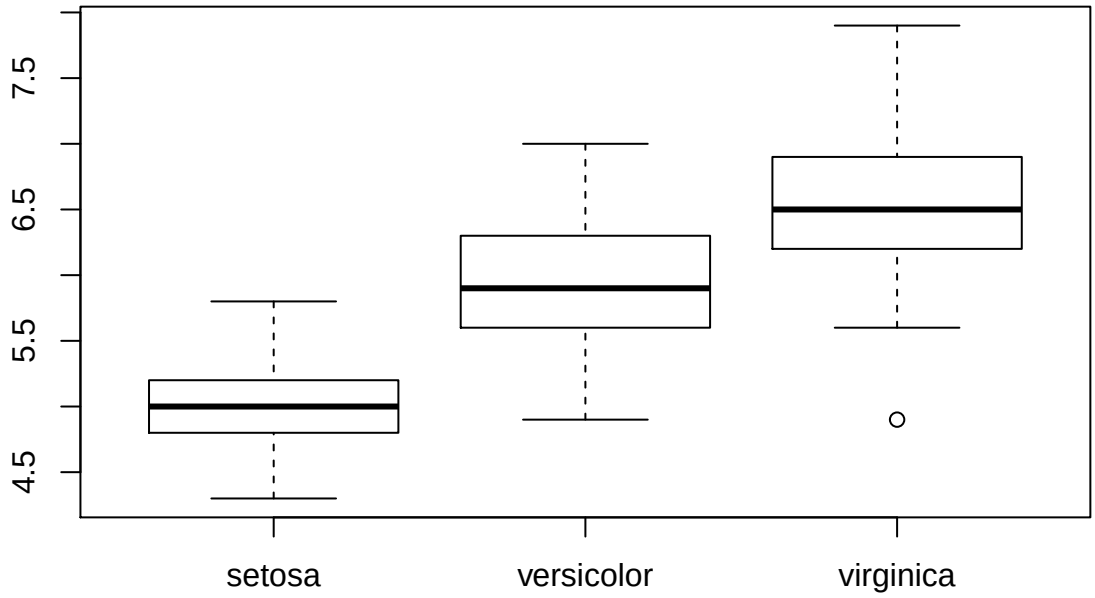
- 3 Distribution in subsets

```
aggregate(Sepal.Length ~ Species, summary, data=iris)
```

```
##      Species Sepal.Length.Min. Sepal.Length.1st Qu. Sepal.Length.Median
## 1     setosa             4.300                4.800                5.000
## 2 versicolor             4.900                5.600                5.900
## 3  virginica             4.900                6.225                6.500
##   Sepal.Length.Mean Sepal.Length.3rd Qu. Sepal.Length.Max.
## 1             5.006                5.200             5.800
## 2             5.936                6.300             7.000
## 3             6.588                6.900             7.900
```

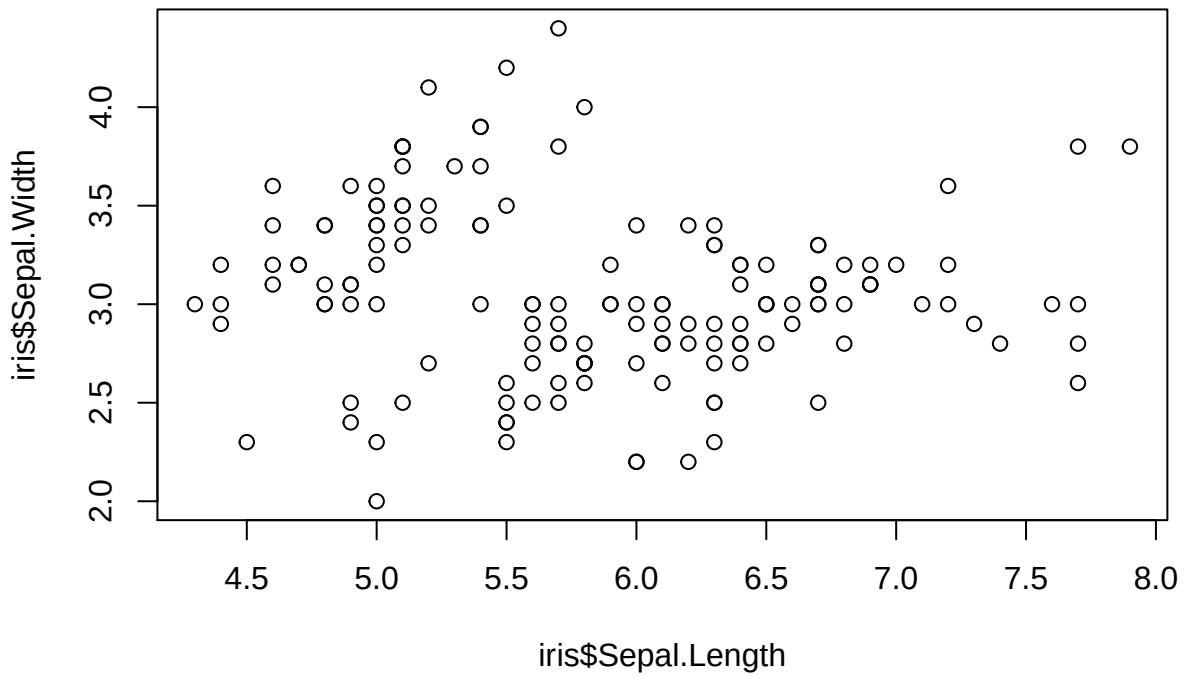
- 4 Box Plot

```
boxplot(Sepal.Length~Species, data=iris)
```



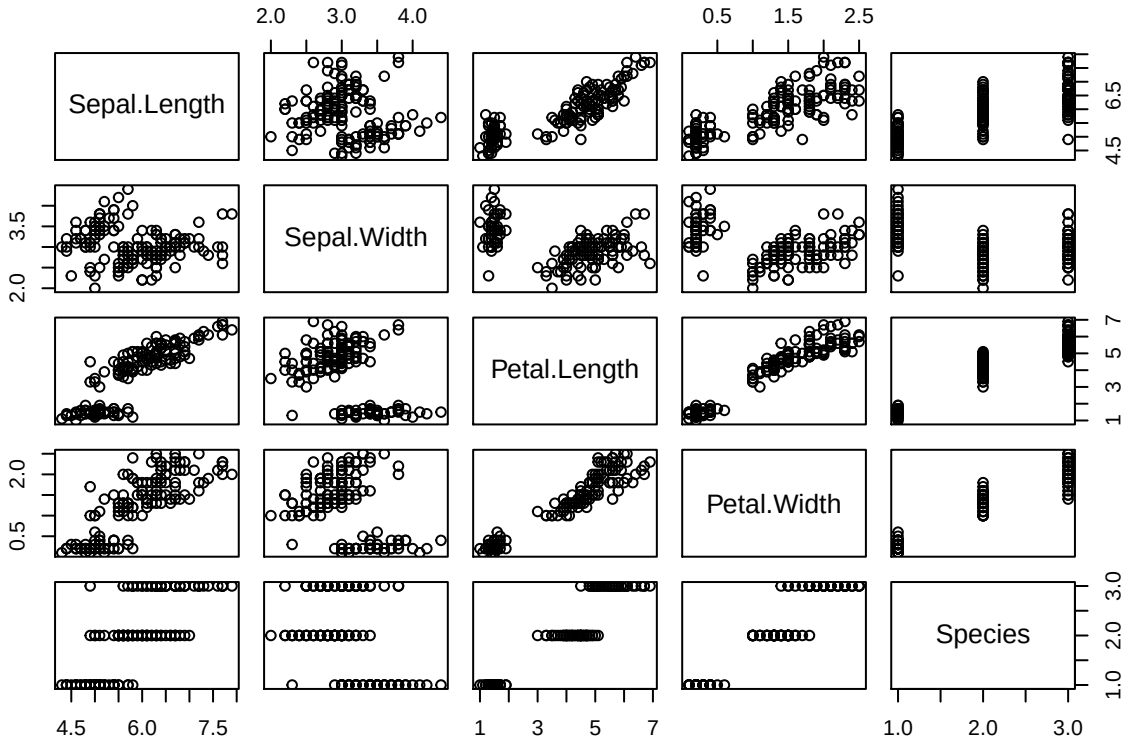
* 5 Scatter plot

```
plot(iris$Sepal.Length, iris$Sepal.Width)
```



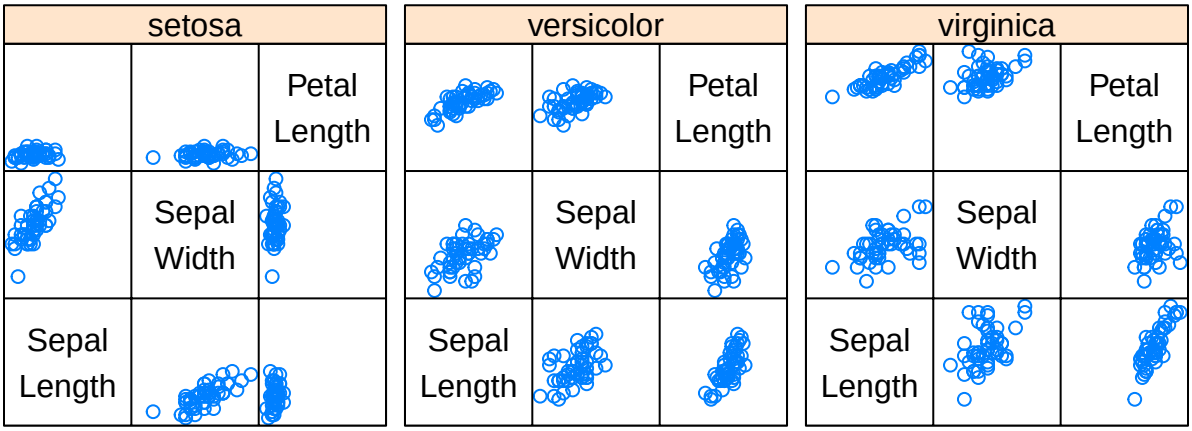
* 6 Pairs plot

```
pairs(iris)
```



* 7 other complicated plots

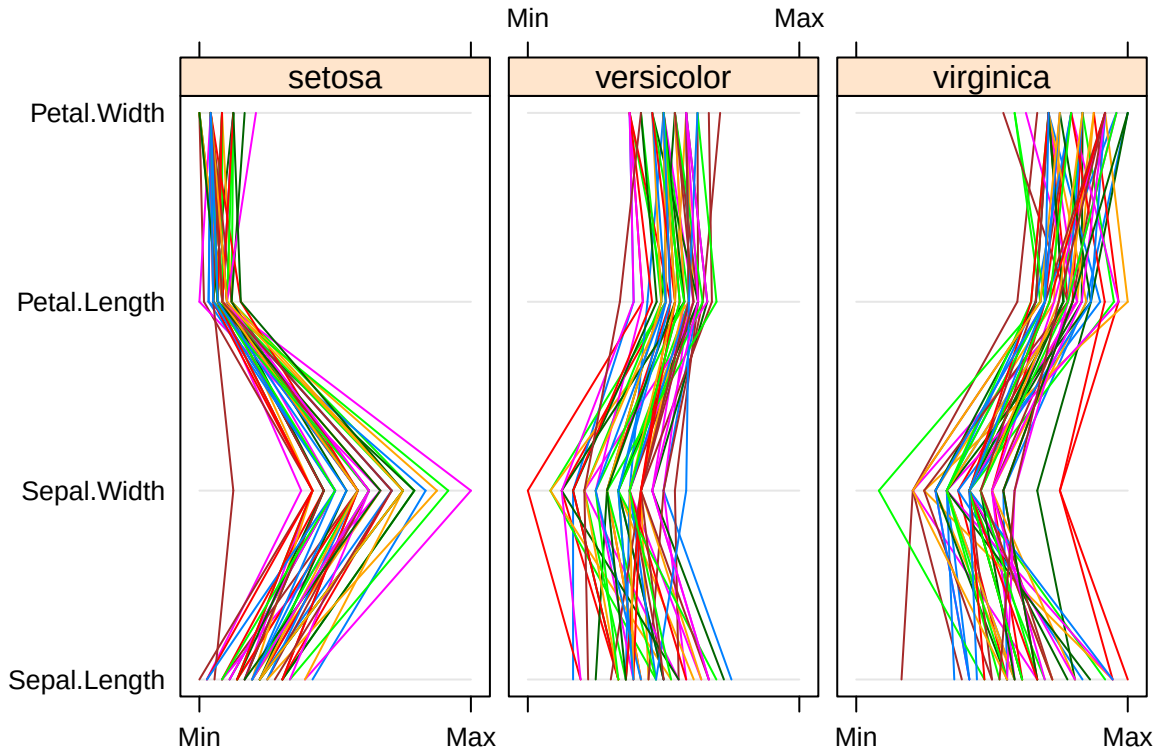
```
library(lattice)
splom(~iris[1:3] | Species, data = iris, pscales = 0,varnames = c("Sepal\nLength", "Sepal\nWidth", "Peta
```



Scatter Plot Matrix

```
#
parallel(~iris[, 1:4] | Species, data = iris, layout = c(3, 1))
```

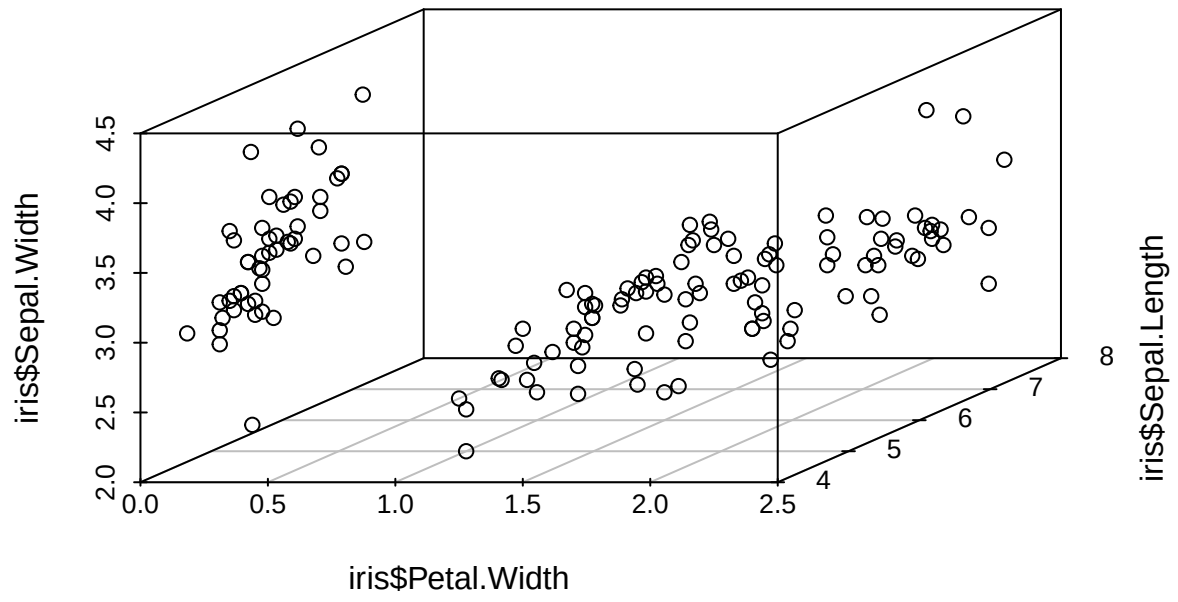
```
## Warning: 'parallel' is deprecated.
## Use 'parallelplot' instead.
## See help("Deprecated")
```



*** ## More Exploration

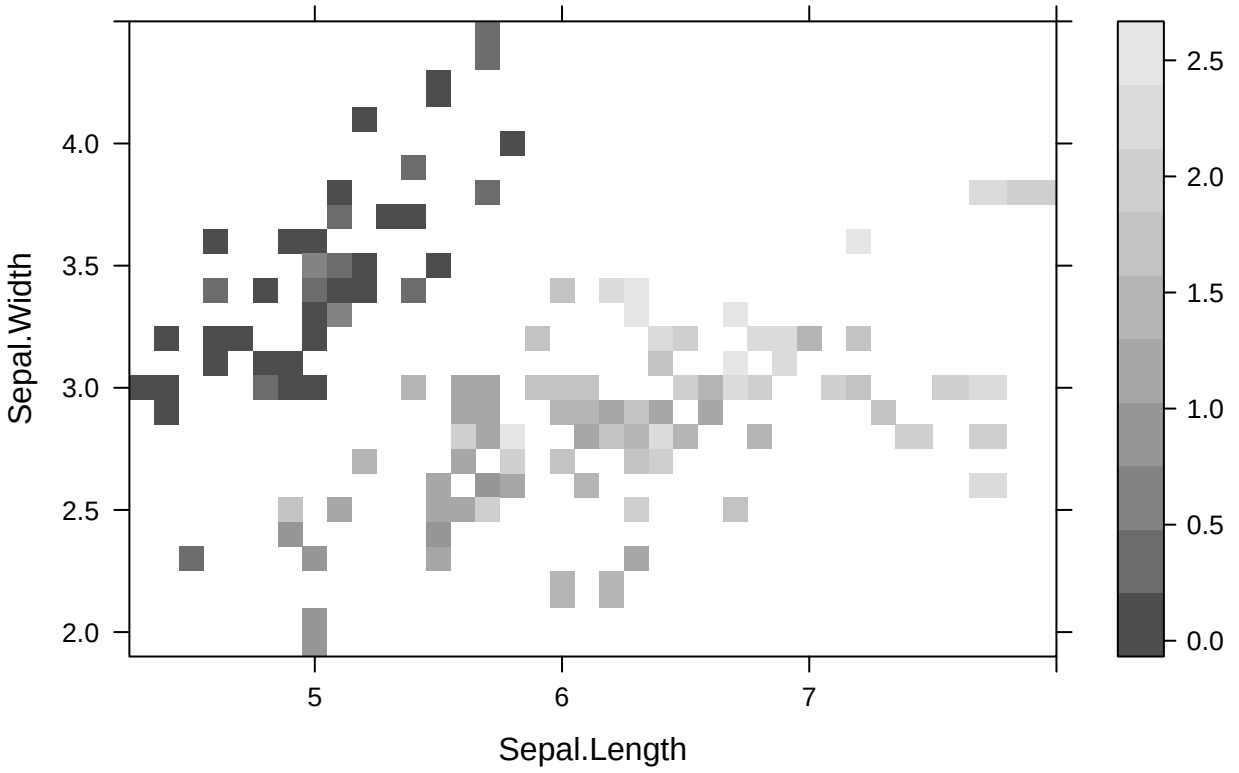
- 3D Scatter plot

```
library(scatterplot3d)
scatterplot3d(iris$Petal.Width, iris$Sepal.Length, iris$Sepal.Width)
```



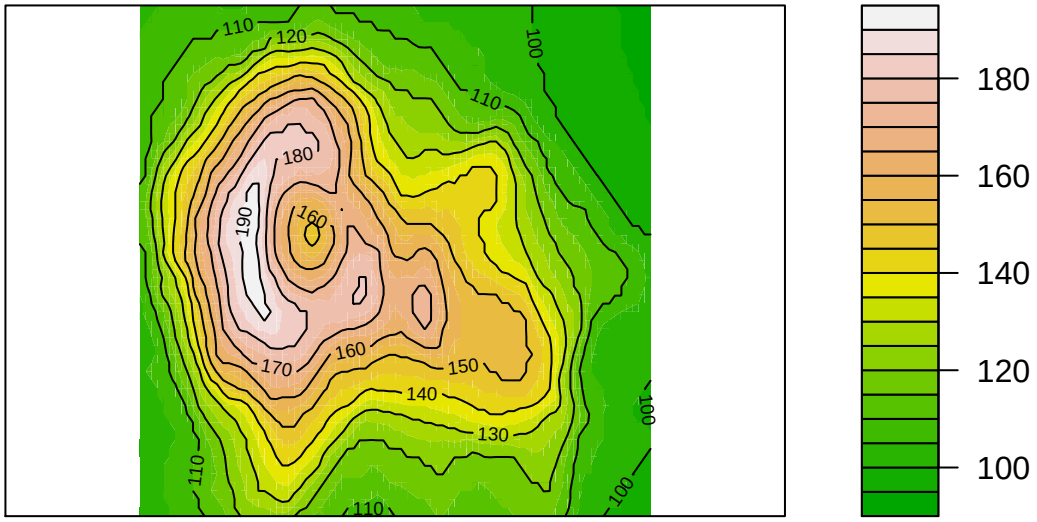
* Level Plot

```
library(lattice)
print(levelplot(Petal.Width~Sepal.Length*Sepal.Width, iris, cuts=9, col.regions=grey.colors(10)))
```



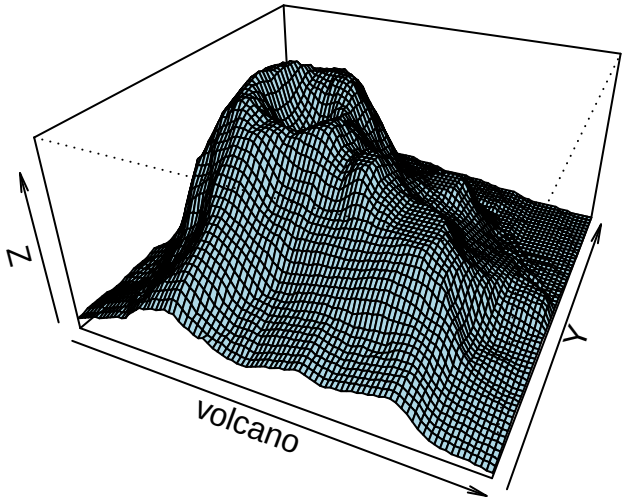
* Contour

```
filled.contour(volcano, color = terrain.colors, asp = 1, plot.axes=contour(volcano, add=T) )
```



* 3D Surface

```
persp(volcano, theta = 25, phi = 30, expand = 0.5, col = "lightblue")
```



* Interactive 3D Scatter Plot

```
library(rgl)
plot3d(iris$Petal.Width, iris$Sepal.Length, iris$Sepal.Width)
```

Writing plots as pdf/ps

- Save as a .PDF file

```
pdf("myPlot.pdf")
x <- 1:50
plot(x, log(x))
graphics.off()
```

- Save as a postscript file

```
postscript("myPlot.ps")
x <- -20:20
plot(x, x^2)
graphics.off()
```

- Find temp.. or save as png or jpg

```r
jpeg("plot.jpg")
plot(x, 1/x)
dev.off()
```

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