Statistical foundations of Machine Learning
INFO-F-422
Introduction to R

Yann-Aël Le Borgne, Fabrizio Carcillo and Gianluca Bontempi

February 24, 2017

I Basics

I.1 How to start and end R

- Start R: type “R” (without the quotes) in a terminal.
- Type `q()` for ending your session.

I.2 Use R as calculator

```
2 + 2
```

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

```
exp(-2)
```

```
## [1] 0.1353353
```

```
rnorm(15)  # to generate random numbers
```

```
##  [1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.12928774
##  [6]  1.71506499  0.46091621 -1.26506123 -0.68685285 -0.44566197
## [11]  1.22408180  0.35981383  0.40077145  0.11068272 -0.55584113
```

I.3 Assign values to variables

```
x <- 2
x
```

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

```
y <- x + x
```

I.4 Installing and loading libraries

```
# install.packages('ISwR') uncomment previous line to install package
library(ISwR)
```

I.5 Naming conventions

- The variables are case-sensitive, can contain a dot (.) and numbers (but cannot start with a number or with a dot followed by a number).
- The names `c,q,t,C,D,F,I,T,diff,df,pt` in particular, are already used by the system and should therefore be avoided.

II Vectors and matrices

II.1 Defining a vector

Vectors can be of any type: integer, double, character, logical, resulting from relational expressions:

```
w <- c(1, 2, 3, 4)
x <- c(1.5, 2.5, 3.5, 3.5)
y <- c("Huey", "Dewey", "Loui")
z <- c(T, F, F, T)
wx <- w > x
wx

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

There exist different functions to define a vector: element wise $c(val_1, \ldots, val_n)$, as a sequence between two values $seq(val_1, val_2)$ and as n repetitions of the same element $rep(val, n)$.

```
w <- c(1, 2, 3, 4)
w

## [1] 1 2 3 4

x <- seq(1, 4)
x

## [1] 1 2 3 4

y <- rep(1, 4)
y

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

Exercises

Generate the following vectors using only the functions `>rep` and `>seq` (no `>c(..)`), check the help by typing `>?rep` and `>?seq` for all parameter options.

```
## [1] 1 3 5
## [1] 1 2 2 2 3 3 3 3 3
```

II.2 Vector arithmetics

```
weight <- c(60, 72, 57, 90, 95, 72, 60)
height <- c(1.75, 1.8, 1.65, 1.9, 1.74, 1.91, 1.69)
bmi <- weight/height^2
bmi

## [1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630 21.00767
```

Exercises

- Compute the adjusted BMI using the formula $bmi_2 = \frac{weight}{height^{2.5}/1.3}$ and store it in a variable `>bmi2`
- What can you conclude?

II.3 Functions for vectors

There exist a lot of predefined functions for a vector v, these include the following

```
v <- seq(5, 1)
sum(v)

## [1] 15

length(v)

## [1] 5

sort(v)

## [1] 1 2 3 4 5

order(v)

## [1] 5 4 3 2 1

prod(v)

## [1] 120

mean(v)

## [1] 3

sd(v)  # standard deviation

## [1] 1.581139
```

II.4 Defining matrices

- In R, matrices (2 dimensions) and arrays (more than 2 dimensions) can contain elements of any type (characters, etc.)
- The `dim` assignment function sets or changes the *dimension attribute* of a vector `x`, causing R to treat the vector as a matrix.

```
x <- 1:12
dim(x) <- c(3, 4)
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

- Other possibilities to create matrices using the `matrix(...)` command:

```
matrix(1:12, nr = 3, byrow = T)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

```
matrix(1, 6, 7)
```

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

- Using the `array` command:

```
array(1:12, dim = c(3, 4), dimnames = list(c("a", "b", "c"), seq(1, 4)))
```

```
##   1 2 3  4
## a 1 4 7 10
## b 2 5 8 11
## c 3 6 9 12
```

- Renaming of rows and transpose of a matrix

```
x <- matrix(1:12, nr = 3, byrow = T)
rownames(x) <- LETTERS[1:3]
x
```

```
##   [,1] [,2] [,3] [,4]
## A    1    2    3    4
## B    5    6    7    8
## C    9   10   11   12
```

- "Glueing" vectors together: functions `rbind()` and `cbind()` (row- and columnwise respectively)

```
cbind(A = 1:4, B = 5:8, C = 9:12)
```

```
##      A B  C
## [1,] 1 5  9
## [2,] 2 6 10
## [3,] 3 7 11
## [4,] 4 8 12
```

```
rbind(A = 1:4, B = 5:8, C = 9:12)
```

```
##   [,1] [,2] [,3] [,4]
## A    1    2    3    4
## B    5    6    7    8
## C    9   10   11   12
```

II.5 Matrix functions

- matrix product, inversion and transpose

```
a <- matrix(1:4, 2, 2, byrow = T)
a
```

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

```
## matrix product
a %*% a
```

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

## inverse matrix
solve(a)

##      [,1] [,2]
## [1,] -2.0  1.0
## [2,]  1.5 -0.5

## transpose of a matrix
t(a)

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

II.6 Indexing

- Allows to select a particular element in a vector, take as example `height`.

```
height

## [1] 1.75 1.80 1.65 1.90 1.74 1.91 1.69
```

- To access the fifth element

```
height[5]

## [1] 1.74
```

- To access several elements in one step

```
height[c(3, 5, 7)]

## [1] 1.65 1.74 1.69

v <- c(3, 5, 7)
height[v]

## [1] 1.65 1.74 1.69
```

- Remove elements from a vector

```
height[-c(3, 5, 7)]

## [1] 1.75 1.80 1.90 1.91
```

- Conditional selection

```
height[height > 1.7]

## [1] 1.75 1.80 1.90 1.74 1.91

height[height > 1.7 & height < 1.9]

## [1] 1.75 1.80 1.74
```

Exercises

Use the `sort` function to sort the vector `height` by decreasing order of weight

```
## [1] 1.74 1.90 1.80 1.91 1.75 1.69 1.65
```

III Factors, lists & data frames

III.1 Factors

- *Factors* are the categorial variables (see `enum` in C++). They make it possible to assign meaningful names to the categories.
- A factor is said to have a set of *levels*.
- Ex: a factor has four levels. It consists of two items: a vector of integers between 1 and 4 and a character vector of length 4 containing strings describing what the four levels are.

```
pain <- c(0, 3, 2, 2, 1)
fpain <- factor(pain, levels = 0:3)
levels(fpain) <- c("non", "mild", "medium", "severe")
```

- `fpain` is a categorial variable.

```
fpain

## [1] non    severe medium medium mild
## Levels: non mild medium severe

as.numeric(fpain)

## [1] 1 4 3 3 2

levels(fpain)

## [1] "non"    "mild"   "medium" "severe"
```

III.2 Lists

- Useful to combine a collection of objects into a larger composite object.

- The following data concern pre- and postmenstrual energy intake in a group of women.

```
intake.pre <- c(5260, 5470, 5640, 6180, 6390, 6515, 6805, 7515, 7515, 8230,
    8770)
intake.post <- c(3910, 4220, 3885, 5160, 5645, 4680, 5265, 5975, 6790, 6900,
    7335)
```

- Combine the vectors into a list

```
mylist <- list(before = intake.pre, after = intake.post)
mylist

## $before
##  [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770
##
## $after
##  [1] 3910 4220 3885 5160 5645 4680 5265 5975 6790 6900 7335
```

- The components of the list are named according to the arguments' names used in *list* (in the example `before` and `after`). Named components may be extracted as follows

```
mylist$before

##  [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770
```

- or alternatively by using the component's index

```
mylist[[1]]

##  [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770
```

III.3 Data frames

- A data frame corresponds to a "data set" in other statistical packages. It is a list of vectors and/or factors of the same length. Data in the same position come from the same experimental unit.
- Creation of a data frame:

```
d <- data.frame(intake.pre, intake.post)
d
```

```
##    intake.pre intake.post
## 1        5260        3910
## 2        5470        4220
## 3        5640        3885
## 4        6180        5160
## 5        6390        5645
## 6        6515        4680
## 7        6805        5265
## 8        7515        5975
## 9        7515        6790
## 10       8230        6900
## 11       8770        7335
```

- Elements in one row correspond to the same woman
- As with lists, variables are accessible using the ”$” notation

```
d$intake.pre

##  [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770
```

IV Loops & conditions

IV.1 Loop types

There are three standard loop types

- `for(name in expr1) instructions` where `expr1` is a vector expression (ex: `1:20`)
- `while(condition) instructions`
- `repeat instructions`

IV.2 Implicit loops

- Apply a function to each element of a set of values or vectors and collect the results in a single structure.
- In R this can be done by using one of the two functions `lapply` and `sapply`. The former always returns a list whereas the latter tries to simplify the result into a vector or matrix if possible.
- The function `apply` allows to apply a function to the rows or columns of a matrix.
- The function `tapply` allows to create tables of the function's value on subgroups defined by its second argument, which can be a factor or a list of factors. (The grouping can also be defined by ordinary vectors. They will be converted to factors internally.)

Exercises

1. Load the "thuesen" data

```
data(thuesen)
```

2. Compute the mean of the data frame `>thuesen` to get the result as a list:

```
## $blood.glucose
## [1] 10.3
##
## $short.velocity
## [1] 1.325652
```

and as a vector:

```
##  blood.glucose short.velocity
##      10.300000       1.325652
```

3. Initialize a matrix, in order to get the same results as in this tutorial, use the same random seed

```
set.seed(12345)
m <- matrix(rnorm(12), 4)
m
```

```
##            [,1]       [,2]       [,3]
## [1,]  0.5855288  0.6058875 -0.2841597
## [2,]  0.7094660 -1.8179560 -0.9193220
## [3,] -0.1093033  0.6300986 -0.1162478
## [4,] -0.4534972 -0.2761841  1.8173120
```

and compute the mean of each row and of each column using `apply`, check the help for details `?apply`. How do you compute the mean of the entire matrix?

```
## [1]  0.3024188 -0.6759373  0.1348491  0.3625436
## [1]  0.1830486 -0.2145385  0.1243956
## [1] 0.03096856
```

IV.3 Conditions

- `if(condition) instructions else if(condition2) instructions2 else instructions3`
- Variants: see documentation.

```
if (m[1, 1] > 0) {
    print(m[1, ])
} else if (m[1, 1] < 0) {
    print(m[, 1])
} else {
    print("element is zero")
}
```

```
## [1]  0.5855288  0.6058875 -0.2841597
```

V Functions

V.1 How to write a basic function

- `name <- function(arg1, arg2, ...) expression`
- Use `return(variable)` to return one variable, multiple variables can be returned as a list.

```
## compute sum of a vector using a for loop
computeSum <- function(x) {
    sum <- 0
    n <- 0
    for (i in 1:length(x)) {
        if (!is.na(x[i])) {
            ## The function is.na is used to determine whether an element is missing
            ## (==NA)
            sum <- sum + x[i]
            n <- n + 1
        }
    }
    # return(sum) will return the sum of the vector
    return(list(sum = sum, mean = sum/n))
}
computeSum(c(1, 2, -4, NA, 6))

## $sum
## [1] 5
##
## $mean
## [1] 1.25
```

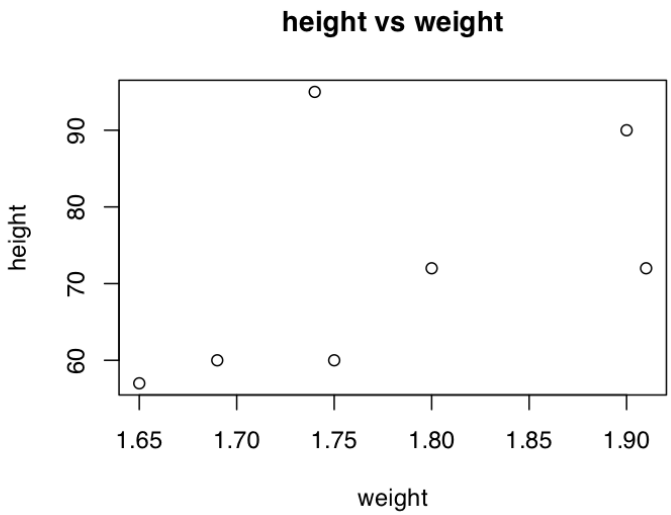
V.2 Compile functions

- A set of functions can be saved in a *.R file
- These functions can be loaded by using the `source(filename)` command

VI Plotting

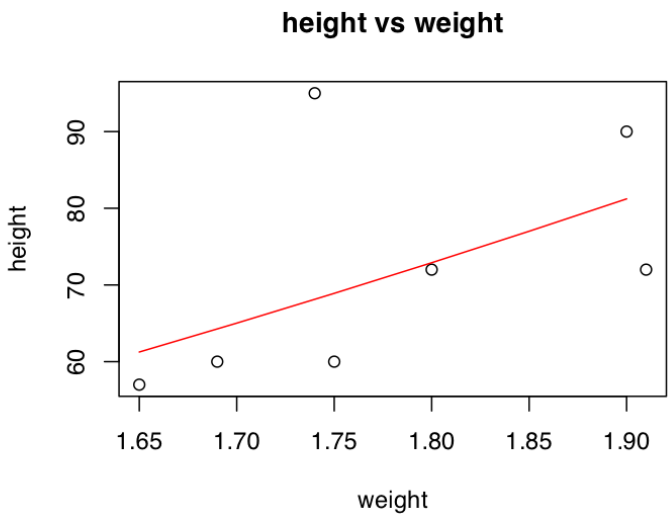
A basic two dimensional plot from using the vectors `>height` and `>weight` can be generated using the `>plot` command.

```
plot(height, weight, main = "height vs weight", xlab = "weight", ylab = "height")
```



Adding pieces to the plot To save a plot to a *.pdf file use the `pdf(...)` command; the

```
plot(height, weight, main = "height vs weight", xlab = "weight", ylab = "height")
hh <- c(1.65, 1.7, 1.75, 1.8, 1.85, 1.9)
lines(hh, 22.5 * hh^2, col = "red")
```



following will save the file `myplot.pdf` in your working directory.

```
pdf(file = "myplot.pdf", width = 5, height = 4)
plot(height, weight, main = "height vs weight", xlab = "weight", ylab = "height")
hh <- c(1.65, 1.7, 1.75, 1.8, 1.85, 1.9)
lines(hh, 22.5 * hh^2, col = "red")
dev.off()
```

VII Probabilities and distributions

VII.1 Sampling

The function `sample` allows to draw random variables uniformly (parameter `prob` to make it non-uniformly) from a vector with or without replacement.

Exercises

- Re-initialize random seed

```
set.seed(123456)
```

- Draw five random variables, uniformly distributed between 1 and 40

```
## [1] 32 30 15 13 14
```

- Simulate 10 coin tosses

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

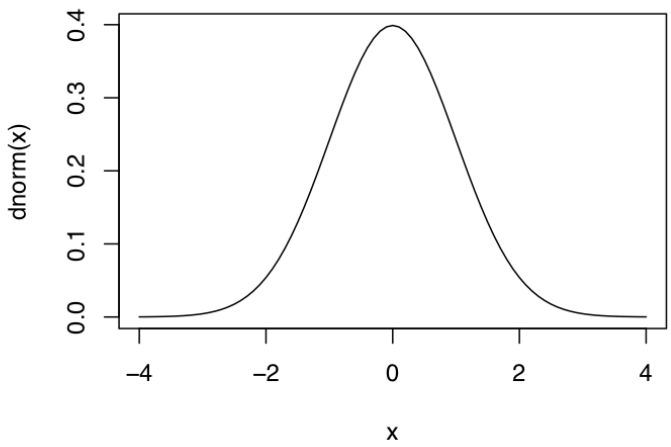
- Sample 10 coin tosses with for a coin in which heads appears with a probability of 0.8

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

VII.2 Distributions in R

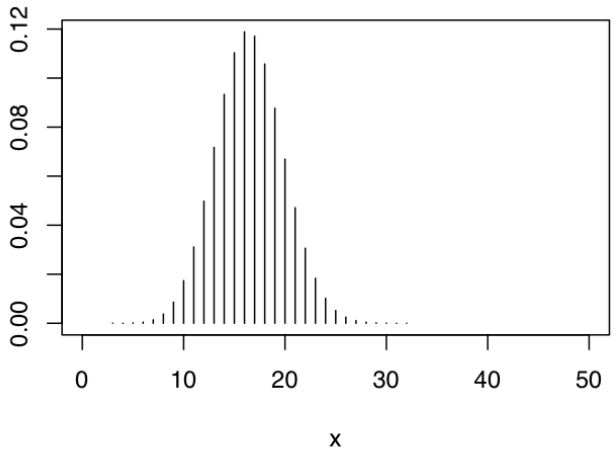
- For a probability distribution, four fundamental values can be computed: the density, the distribution function, the quantiles and the pseudo-random numbers. Using R, each of these can be computed for the included distributions.
- For example for a normal distribution (respectively): `dnorm, pnorm, qnorm` et `rnorm`. The same nomenclature is used for all distributions: the density (or the discrete probability) starts with a “d”, the distribution function with “p”, the quantile with “q” and the random number generating function with “r”.
- Plot normal density

```r
x <- seq(-4, 4, 0.1)
# alternatively curve(dnorm(x), from=-4, to=4)
plot(x, dnorm(x), type = "l")
```



Exercises

- Plot the density for a binomial random variable $B(50, 0.33)$ (use the parameter `type="h"` for the plot function to generate a pin diagram)



- Compute the probability of a normally distributed variable with mean 132 and standard deviation 13 being smaller than 160

```
## [1] 0.9843739
```

VII.3 Quantiles

- Definition: the quantile is the inverse of the distribution function. The p–quantile is by definition that value having the property that there is a probability p to obtain a value lower or equal to it. For example, the median is the 50% quantile.
- In order to compute the quantiles of an empirical distribution for which samples are given in a vector, we use the function `quantile` which takes as argument this vector and the probability for which the quantile is needed.
- The quantiles are used for computing the confidence intervals. Let n observations be drawn from a normal distribution with the same mean μ and the same standard-deviation σ. It is well known that the observed mean $\overline{x}$ follows a normal distribution

with mean μ and standard deviation $\sigma/\sqrt{n}$. A confidance interval of 95% for μ can be obtained by

$$\overline{x} + \sigma/\sqrt{n} \times N_{0.025} \leq \mu \leq \overline{x} + \sigma/\sqrt{n} \times N_{0.975}$$

where $N_{0.025}$ is the 2.5% quantile of the normal distribution.

Exercises

Let $\sigma = 12$ and $n = 5$ with a mean of $\overline{x} = 83$, compute the confidence interval using the normal distribution

```
## [1] 72.48173
## [1] 93.51827
```

VII.4 Generation of pseudo-random numbers

- The first argument of a function generating random numbers is always the number of samples that should be generated. The next arguments specify the parameters fo the chosen distribution (mean, etc.) Some examples are provided to present "typical" choices of these parameters.

```
rnorm(10)

##  [1]  0.5258529  1.1640281 -0.9701434  1.1222795 -1.1795938 -1.4032109
##  [7]  1.1340698  0.2673613  0.6420977 -0.5403923

rnorm(10, mean = 7, sd = 5)

##  [1]  6.177256 10.231478  5.234626 10.354470 12.452628 16.019931 11.911763
##  [8]  2.345110 -3.312488  2.454138

rbinom(10, size = 20, prob = 0.5)

##  [1] 11  4  9  7  5  8 11 13  9 12
```

- The algorithms for generating pseudo-random numbers are initialised by a seed which we can define in R using the function `set.seed(init.value)`, where `init.value` is an integer. This allows the user to reproduce computations with the same pseudo-random numbers.

VII.5 Table of distributions in R

- The following table provides an overview over the included distributions and their parameters.

Distribution	R name	additional arguments
beta	`beta`	`shape1, shape2, ncp`
binomial	`binom`	`size, prob`
Cauchy	`cauchy`	`location, scale`
chi-squared	`chisq`	`df, ncp`
exponential	`exp`	`rate`
F	`f`	`df1, df2, ncp`
gamma	`gamma`	`shape, scale`
geometric	`geom`	`prob`
hypergeometric	`hyper`	`m, n, k`
log-normal	`lnorm`	`meanlog, sdlog`
logistic	`logis`	`location, scale`
negative binomial	`nbinom`	`size, prob`
normal	`norm`	`mean, sd`
Poisson	`pois`	`lambda`
Student's t	`t`	`df, ncp`
uniform	`unif`	`min, max`
Weibull	`weibull`	`shape, scale`
Wilcoxon	`wilcox`	`m, n`

VII.6 Descriptive statistics

- Computation of the mean, the variance, the standard deviation and the median:

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

## [1] -0.186168

var(x)  ## variance

## [1] 1.014434

sd(x)  ## standard deviation

## [1] 1.007191

median(x)

## [1] -0.1526649
```

Exercises

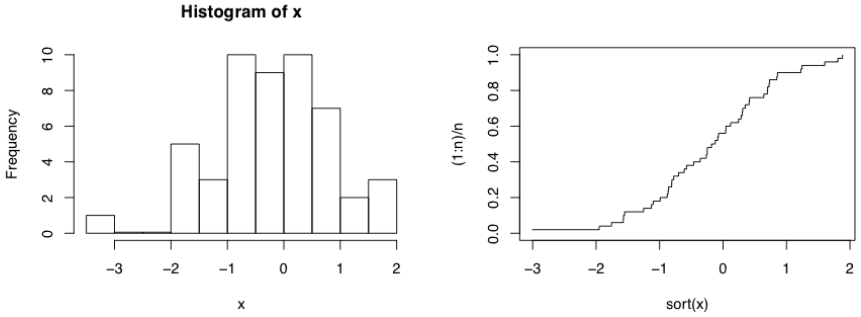
- Generate the following quantiles using `x` and the `quantile()` function

```
##          0%         10%         20%         30%         40%         50%
## -3.00878868 -1.55296713 -0.90168640 -0.78633773 -0.40114554 -0.15266488
##         60%         70%         80%         90%        100%
##  0.07514433  0.32140434  0.69955651  0.89513609  1.88284173
```

VII.6.1 Plots of distributions

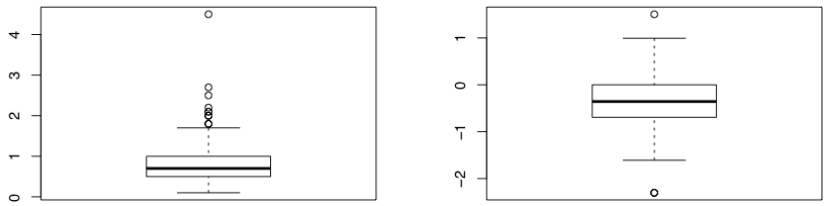
- Histogram and empirical cumulative distribution (the empirical distribution function is defined as the number of data points smaller or equal to x devided by the total number of points)

```
par(mfrow = c(1, 2))
hist(x)
n <- length(x)
plot(sort(x), (1:n)/n, type = "s", ylim = c(0, 1))
```



- In a boxplot, the box in the middle of the graph indicates the quartiles and the median.The two horizontal lines represent the largest (or the smallest) observation which falls within a distance of 1.5 times the size of the box. The observations outside this box are considered as ”extremes” and are noted by points.

```
data(IgM)
par(mfrow = c(1, 2))
boxplot(IgM)
boxplot(log(IgM))
```

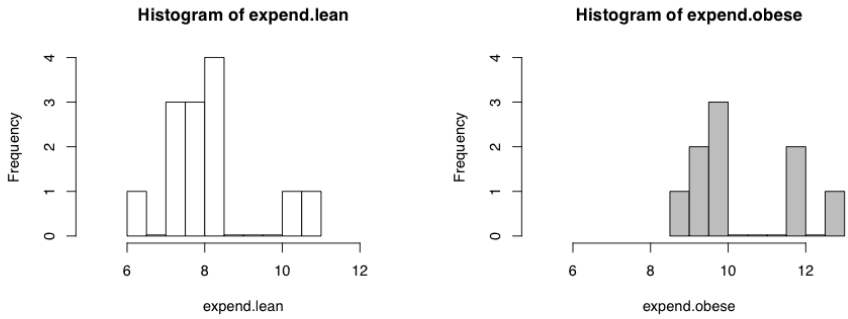


VII.6.2 Graphs for grouped data

We are presenting here some techniques for comparing plots between groups.

Histograms

```
data(energy)
attach(energy)
expend.lean <- expend[stature == "lean"]
expend.obese <- expend[stature == "obese"]
par(mfrow = c(1, 2))
hist(expend.lean, breaks = 10, xlim = c(5, 13), ylim = c(0, 4), col = "white")
hist(expend.obese, breaks = 10, xlim = c(5, 13), ylim = c(0, 4), col = "grey")
```

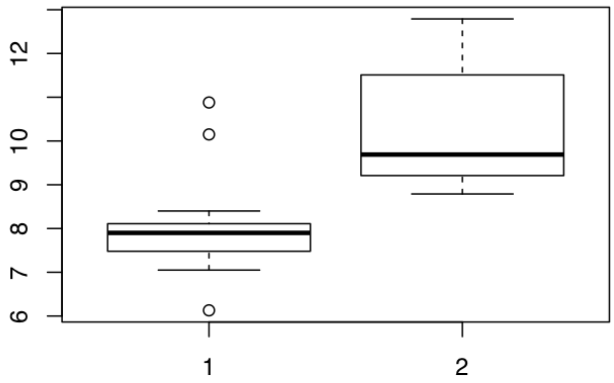



```
par(mfrow = c(1, 1))
```

Parallel boxplots

It is possible to insert boxplots of several different groups in the same window

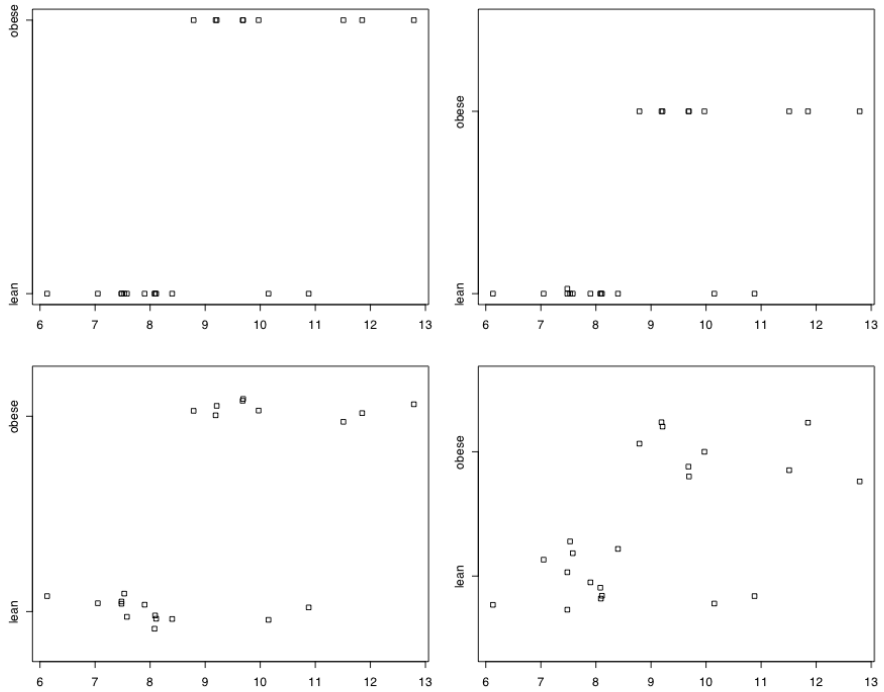
```
boxplot(expend.lean, expend.obese)
```



Stripcharts

- The stripcharts provide the possibility to display the raw data in a graph. We present four variants:

```
opar <- par(mfrow = c(2, 2), mex = 0.8, mar = c(3, 3, 2, 1) + 0.1)
stripchart(expend ~ stature)
stripchart(expend ~ stature, method = "stack")
stripchart(expend ~ stature, method = "jitter")
stripchart(expend ~ stature, method = "jitter", jitter = 0.3)
```




```
par(mfrow = c(1, 1))
```

- The first version superimposes identical points. The parameter `stack` allows to stack identical data vertically. The parameter `jitter` allows to apply a random vertical lag to each point. The last version applies a random lag smaller than the one in the previous version.

Generate tables

- A bi-dimensional table can be generated using `matrix`. The following example concerns the caffeine consumption while giving birth, with respect to women's civil status.

```
caff.marital <- matrix(c(652, 1537, 598, 242, 36, 46, 38, 21, 218, 327, 106,
    67), nrow = 3, byrow = T)
caff.marital

##      [,1] [,2] [,3] [,4]
## [1,]  652 1537  598  242
## [2,]   36   46   38   21
## [3,]  218  327  106   67
```

- Adding names to the rows and columns:

```
colnames(caff.marital) <- c("0", "1-150", "151-300", ">300")
rownames(caff.marital) <- c("Married", "Prev.married", "Single")
caff.marital

##                0 1-150 151-300 >300
## Married      652  1537     598  242
## Prev.married  36    46      38   21
## Single       218   327     106   67
```

- In general, we want to create tables from a data set. In this case, we have to use the functions `table`, `xtabs`, or `ftable`. The factors' levels are directly used as the columns' names. The function `table` is the simplest one of the three.

```
data(juul)
attach(juul)
juul$sex <- factor(juul$sex, labels = c("M", "F"))
juul$menarche <- factor(juul$menarche, labels = c("No", "Yes"))
juul$tanner <- factor(juul$tanner, labels = c("I", "II", "III", "IV", "V"))
```

- Then the respective tables are

```
table(sex)

## sex
##   1   2
## 621 713

table(sex, menarche)
```

```
##    menarche
## sex   1   2
##   1   0   0
##   2 369 335

table(menarche, tanner)

##         tanner
## menarche   1   2   3   4   5
##        1 221  43  32  14   2
##        2   1   1   5  26 202
```

- The variable `menarche` explains that a woman already had her first period and the variable `tanner` provides the state of puberty.

Exercises: Marginal tables and relative frequencies

- Define the following table

```
tanner.sex = table(tanner, sex)
tanner.sex

##       sex
## tanner   1   2
##      1 291 224
##      2  55  48
##      3  34  38
##      4  41  40
##      5 124 204
```

- Compute the marginal tables, i.e. the sum of values with respect to one of the other table's dimensions using the function `margin.table()`

```
## tanner
##   1   2   3   4   5
## 515 103  72  81 328
## sex
##   1   2
## 545 554
```

- Using the function `prop.table()`, compute the table of relative frequencies can be computed. This contains the relative frequencies that are the proportions of the row's or column's total.

```
##       sex
## tanner         1         2
##      1 0.5650485 0.4349515
##      2 0.5339806 0.4660194
##      3 0.4722222 0.5277778
##      4 0.5061728 0.4938272
##      5 0.3780488 0.6219512
```

- Compute the proportions with respect to the global total of the table:

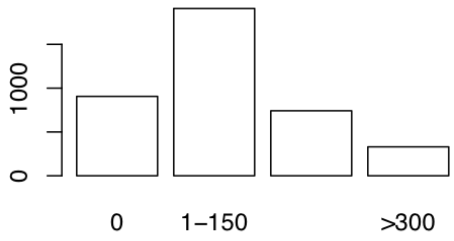
```
##       sex
## tanner          1          2
##      1 0.26478617 0.20382166
##      2 0.05004550 0.04367607
##      3 0.03093722 0.03457689
##      4 0.03730664 0.03639672
##      5 0.11282985 0.18562329
```

VII.6.3 Graphical display of the tables

Bar plots

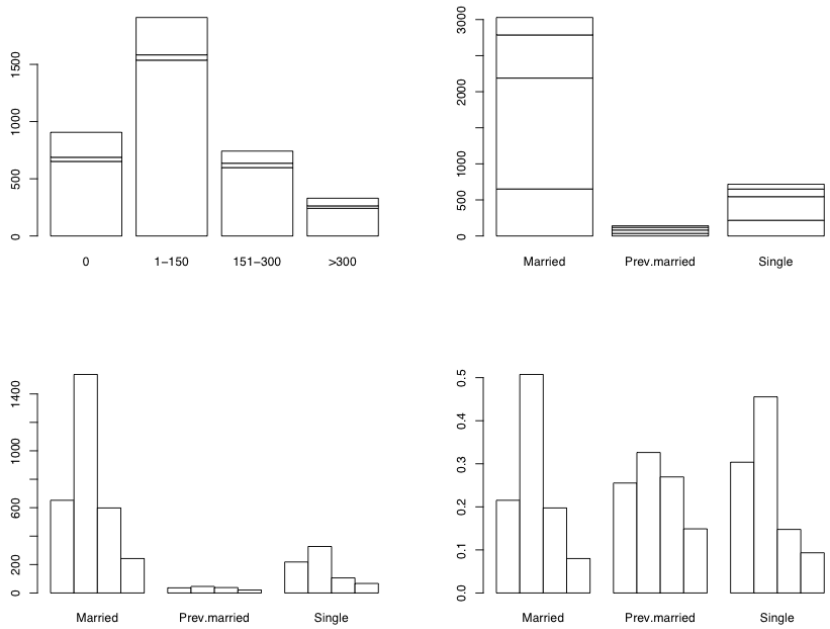
- Returning to the caffein example.
- The function `barplot` takes a vector or a matrix as argument. The most basic version:

```
total.caff <- margin.table(caff.marital, 2)
barplot(total.caff, col = "white")
```



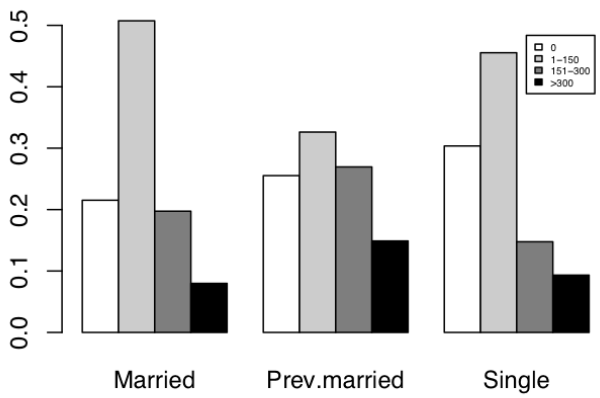
- If the argument is a matrix then `barplot` creates a stacked box plot in which the columns are partitioned with respect to the contributions of the table's different rows. For displaying side-by-side the contributions of a row instead of stacking them, use the parameter `beside=T`. Some examples:

```
par(mfrow = c(2, 2))
barplot(caff.marital, col = "white")
barplot(t(caff.marital), col = "white")
barplot(t(caff.marital), col = "white", beside = T)
barplot(prop.table(t(caff.marital), 2), col = "white", beside = T)
```



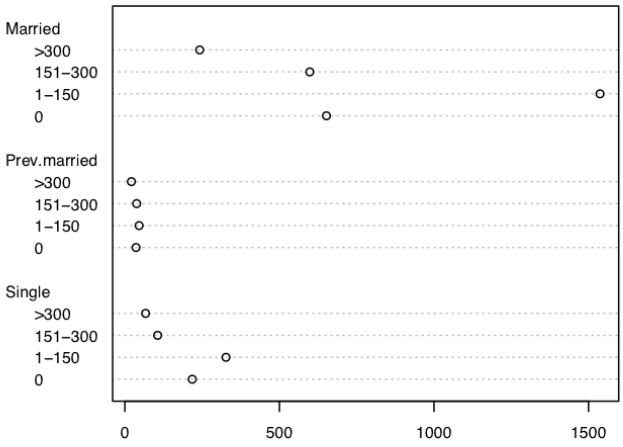
- We can make the plots more beautiful

```
par(mfrow = c(1, 1))
barplot(prop.table(t(caff.marital), 2), beside = T, legend.text = colnames(caff.marital),
    col = c("white", "grey80", "grey50", "black"), args.legend = c(cex = 0.4))
```

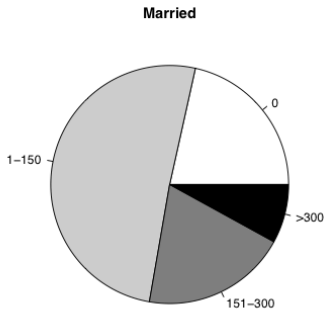


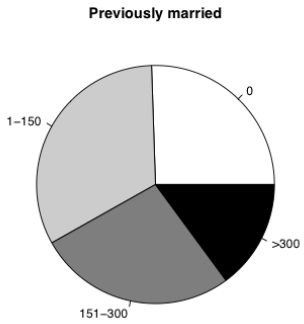
Exercises

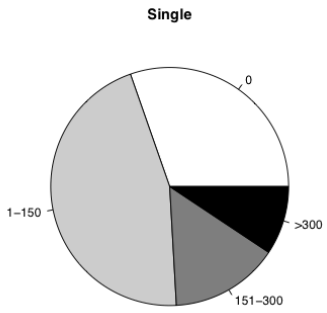
- Create a dotchart with the same information as the previous bar plots (use the command `dotchart()`).



- In order to represent the caffeine consumption table in function of the civil state create a set of pie charts using the function `pie()`.


Married
0
1–150
>300
151–300


Previously married
0
1–150
>300
151–300


Single
0
1–150
>300
151–300

Session Info

- R version 3.3.2 (2016-10-31), `x86_64-apple-darwin13.4.0`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: ISwR 2.0-7, knitr 1.15.1
- Loaded via a namespace (and not attached): evaluate 0.10, formatR 1.4, highr 0.6, magrittr 1.5, stringi 1.1.2, stringr 1.2.0, tools 3.3.2