Statistical Programming — Worksheet 3

1. Cystic Fibrosis dataset

On the class website you will find the file `cystfibr.txt` which contains a set of measurements on a set of individuals with cystic fibrosis. Save the file locally onto your computer.

(a) Take a look at this file using a text editor like Wordpad or Notepad. Read the data into R using `read.table()`; call the resulting dataframe `cf`. *You need* ***`header=TRUE`*** *to get this to work:*

```
> cf = read.table("cystfibr.txt", header = TRUE)
```

Check that the data have been formatted correctly by typing:

```
> head(cf)
```

The first two rows will look wrong if you failed to use ***`header=TRUE`****.*

(b) Obtain the following:

(i) the number of individuals in the dataset;

```
> nrow(cf)
## [1] 25
```

(ii) the number of variables measured on each individual;

```
> ncol(cf)
## [1] 10
```

(iii) a vector of the names of the variables measured on each individual;

```
> names(cf)
##  [1] "age"    "sex"    "height" "weight" "bmp"    "fev1"   "rv"
##  [8] "frc"    "tlc"    "pemax"
```

or: ***`colnames(cf)`***

(iv) the mean, median, standard deviation and range of each of the variables (use `apply()`);

```
> apply(cf, 2, mean)
> apply(cf, 2, median)
> apply(cf, 2, sd)
> apply(cf, 2, range)
```

(c) Create the following data frames and for each one calculate the mean of each of the variables

(i) a new data frame containing just individuals older than 15

```
> cf2 <- cf[cf$age > 15, ]
> lapply(cf2, mean)  # or use apply() as in previous question
```

(ii) a new data frame containing just individuals with bmp in the interval [70,90]

```
> cf[cf$bmp >= 70 & cf$bmp <= 90, ]
> subset(cf, bmp >= 70 & bmp <= 90)  # an alternative
```

(iii) a new data frame containing just individuals with either FEV1 greater than 30 or RV greater than 300.

```
> cf[cf$fev1 > 30 | cf$rv > 300, ]
```

2. Data Frames

Load the `MASS` library and take a look at the dataset called `survey`.

```
> library(MASS)
> head(survey)
```

You can look at the documentation:

```
> ?survey
```

and get a brief summary of each variable:

```
> summary(survey)
```

(a) Find the mean pulse rate of the students. What goes wrong here? *There are missing values, so*

```
> mean(survey$Pulse)
```

leads to `NA`, since logically the mean cannot be determined without knowing all the values.

The vector of pulses stored in `survey` contains some entries which are labelled `NA`. This is used in `R` to represent **missing data**.

(b) Try looking at the documentation for `mean()` to see how to get around this.
Inspection of the documentation shows that you can get around this by using

```
> mean(survey$Pulse, na.rm = TRUE)

## [1] 74.15104
```

which ignores missing values for the purposes of calculating the mean.

(c) The ages are recorded as fractions representing a number of months. Change that columns of the data frame so that it contains whole years (the `floor()` command may be useful).

```
> survey$Age = floor(survey$Age)
```

(d) Find the mean pulse rate for students under 20.

```
> with(survey, mean(Pulse[Age < 20], na.rm = TRUE))

## [1] 75.48551
```

Subsetting. Suppose I want to obtain the records of students who are over 190 cm tall. Since data frames allow me to subset just like a matrix, I might just think of typing:

```
> survey[survey$Height > 190, ]
```

What goes wrong here? *Every missing height value gives to an* `NA` *in the logical vector we're subsetting with. This leads (in each case) to a row of* `NA`*s in the data frame.* Try instead the following:

```
> subset(survey, Height > 190)
```

The subset function ignores missing values, which is usually the behaviour we would prefer. We can also select only some of the fields of the data frame if we prefer:

```
> subset(survey, Height > 190, select = c("Pulse", "Clap"))
```

Recall that the `&` operator does a point-wise logical 'and' comparison.

```
> subset(survey, (Pulse > 70) & (Smoke == "Heavy"))
```

Similarly, `|` is for 'or', and `!` for 'not'.

```
> with(survey, (Pulse > 70) | (Pulse < 45))
> !(survey$Age > 30)
```

(e) Find the mean age of students who write with their right hand. *Either of:*

```
> mean(subset(survey, W.Hnd == "Right")$Age)
> mean(survey$Age[survey$W.Hnd == "Right"], na.rm = TRUE)
```

In the second case the `na.rm=TRUE` *is necessary even though no ages are missing, because the handedness of one student is missing.*

(f) What proportion of left handers do not clap with their left hand on top?

```
> mean(subset(survey, W.Hnd == "Left", select = Clap) != "Left")

## [1] 0.5

> mean(survey$Clap[survey$W.Hnd == "Left"] != "Left", na.rm = TRUE)

## [1] 0.5
```

(g) Using the `plot()` command, plot the pulse of the subjects against their age.

```
> plot(survey$Age, survey$Pulse)
```

Try subtracting 10 from the age and taking the logarithm (using the function `log()`), to obtain a slightly clearer picture.

```
> with(survey, plot(log(Age - 10), Pulse))
```

3. Factors

Take a look at the `birthwt` data from the `MASS` package.

(a) How is race stored in these data? Is this sensible? *It's stored as a numeric vector, which isn't sensible since these data are categorical.*

(b) Turn this into a factor with level names as indicated in the documentation. To do this, look at the documentation for the function `factor()`. *One way is*

```
> birthwt$race <- factor(birthwt$race, labels = c("white", "black", "other"))
```

but you might also turn it into a character vector first:

```
> birthwt$race <- as.factor(c("white", "black", "other")[birthwt$race])
```

(c) Use the `table()` command to count the number of babies of each race.

```
> table(birthwt$race)

##
## white black other
##    96    26    67
```

(d) Try the following command:

```
> tab = with(birthwt, table(smoke, low))
```

What do the results in `tab` suggest? *Comparing*

```
> tab[1, 2]/sum(tab[1, ])

## [1] 0.2521739

> tab[2, 2]/sum(tab[2, ])

## [1] 0.4054054
```

it seems that a much higher proportion of babies whose mother smoked during pregnancy were of low birthweight (although the documentation doesn't specifiy how the data were collected, so we have to be careful what we conclude from this.)

4. Tables An 18×8 foot area of field planted with maize was split into 3×1 foot rectangles. The `beetlelarva.txt` data record the $n = 48$ counts of Japanese beetle larvae found in the top foot of soil of each rectangle. Load it using the connection

```
> df <- read.table("http://www.stats.ox.ac.uk/~nicholls/PartASSP/beetlelarva.txt")
```

The beetles are spread by the planting and ploughing process. In which direction, `northing` or `easting`, would you say this field was ploughed? In order to answer this question convert the data to a table with

```
> tb <- xtabs(count ~ easting + northing, data = df)
```

Look at `tb` and try plotting `df` and `tb`. Can you guess? *First take a look at the numbers*

```
> str(df)

## 'data.frame': 48 obs. of  3 variables:
##  $ northing: int  6 5 4 3 2 1 6 5 4 3 ...
##  $ easting : Factor w/ 8 levels "A","B","C","D",..: 1 1 1 1 1 1 2 2 2 2 ...
##  $ count   : int  5 9 10 6 7 11 6 8 12 6 ...
```

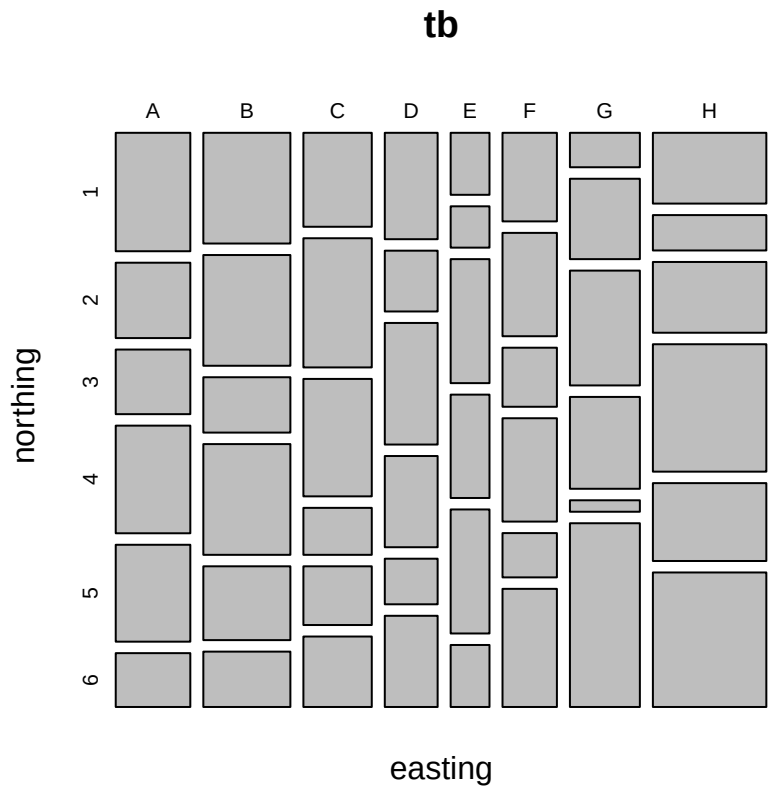
We can see the variables and their types. Looking at `tb` *itself,*

```
> tb
```

```
##         northing
## easting  1  2  3  4  5  6
##       A 11  7  6 10  9  5
##       B 12 12  6 12  8  6
##       C  8 11 10  4  5  6
##       D  7  4  8  6  3  6
##       E  3  2  6  5  6  3
##       F  6  7  4  7  3  8
##       G  3  7 10  8  1 16
##       H 10  5 10 18 11 19
```

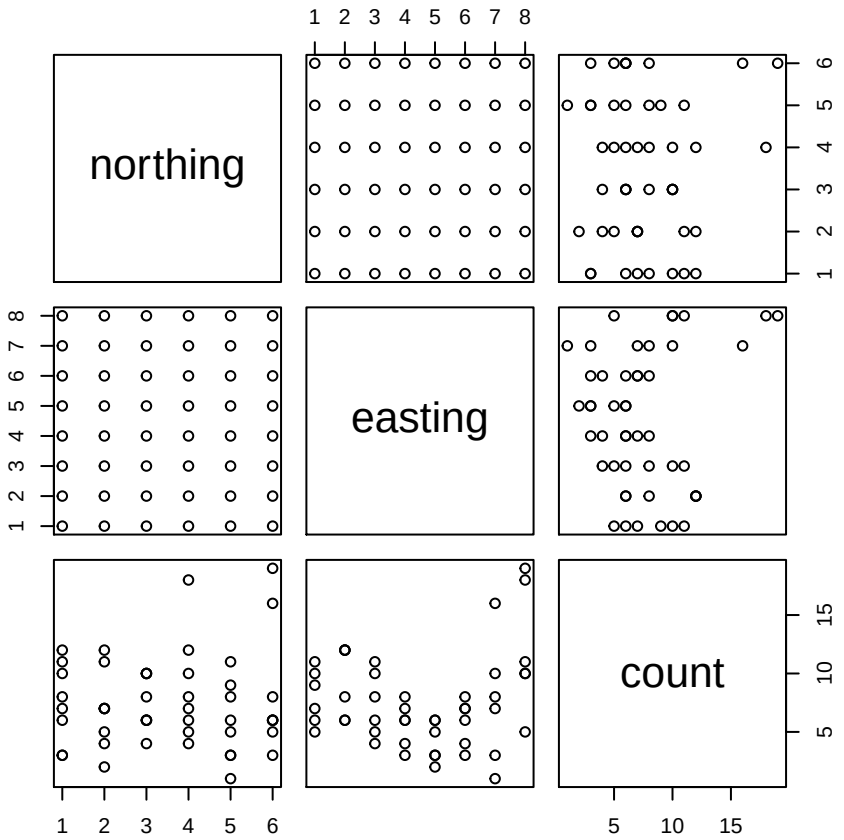
it seems that the easting rows D,E and F are low while rows A,B,C,G and H have higher values. For the northing columns the variation is less clear. If we make a mosaic plot

```
> plot(tb)
```



this relationship is clear (try `plot(t(tb))` *to get the "transpose" plot). We can also plot all the columns of the data frame against one another, like this*

```
> plot(df)
```



Again, the trend of `count` *with* `easting` *is clear, while no clear pattern is discernable for* `northing`*. We would guess the plough ran north-south so that variation north-south is smoothed out, variation east-west is fixed. Note that if we load a table and need to work with a data frame, the reverse command to* `xtabs` *is just* `data.frame()`*.*

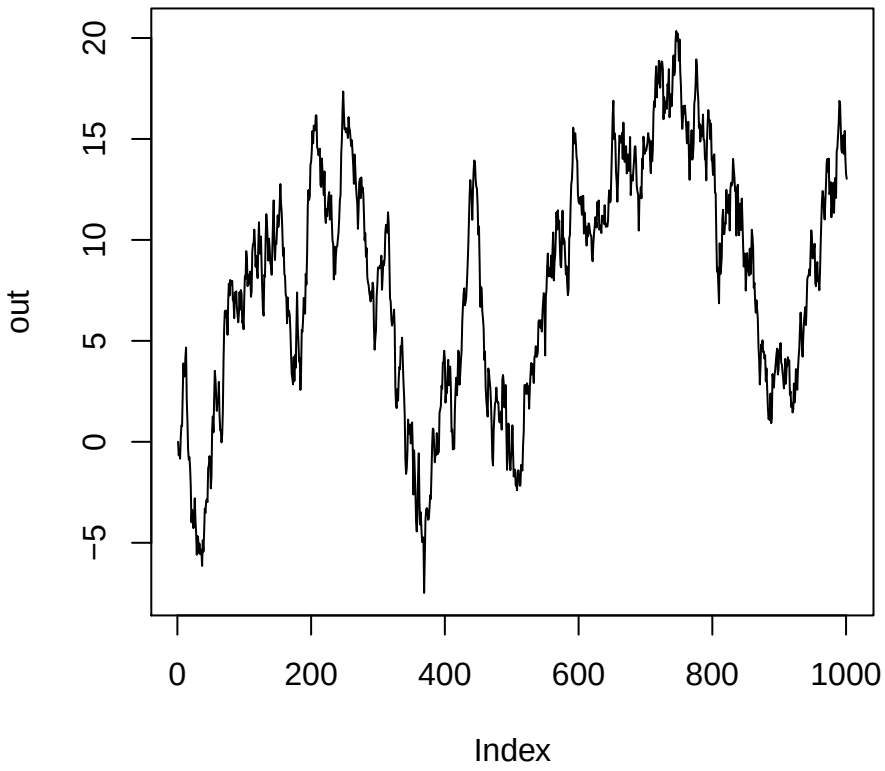
```
> df.from.tb <- data.frame(tb)
> head(df.from.tb)

##   easting northing Freq
## 1       A        1   11
## 2       B        1   12
## 3       C        1    8
## 4       D        1    7
## 5       E        1    3
## 6       F        1    6

> str(df.from.tb)

## 'data.frame': 48 obs. of  3 variables:
##  $ easting : Factor w/ 8 levels "A","B","C","D",..: 1 2 3 4 5 6 7 8 1 2 ...
##  $ northing: Factor w/ 6 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 2 2 ...
##  $ Freq    : int  11 12 8 7 3 6 3 10 7 12 ...
```

5. ***Gaussian Random Walk.** Write a function which simulates a Gaussian random walk with `n` steps. (In other words, $X_0 = 0$ and $X_i - X_{i-1} \sim N(0, 1)$ independently.)

```
> rand_walk <- function(n) {
+     x = 0
+     for (i in 1:n) {
+         x[i + 1] = x[i] + rnorm(1)
+     }
+     x
+ }
```

Generate and plot the walk for `n = 1000` (use `type="l"`).

```
> out <- rand_walk(1000)
> plot(out, type = "l")
```



Try to vectorise your code (if it is already vectorised, try constructing a naïve version with a `for()` loop). Compare the speed of the vectorised and unvectorised versions by generating a random walk of length 50,000

Using `cumsum()` is much faster.

```
> rand_walk2 <- function(n) {
+     x <- c(0, cumsum(rnorm(n)))
```

```
+     x
+ }
```

A major problem in the first version is that we're constantly adding one extra element onto the end of the vector, so R has to create a new object every time.