Log 708 - Chapter 3 Solutions

Halvard Arntzen

These are suggested solutions to chapter 3 exercises. In many cases, R offers different ways of doing things, so this is not a list of "definitive" answers. And of course, even though a major part of these exercises is "watching videos", we do not watch the videos for you :-)

3.1

This is something you just have to do on your own!

3.2

b.

```
#keyboard shortcut CTRL+ENTER   (Used with the script editor)
Executes code currently at the cursor (Or selected part of code)

#keyboard shortcut CTRL+C
Copies selected code. Very useful.

#keyboard shortcut CTRL+V
Pastes copied code. Equally useful.

#keyboard shortcut CTRL+Z (works in the editor only)
UNDO previous keystrokes. Works sequentially backwards to undo previous
steps. (Actually works in the console to some extent also.)


#Arrow up / down (works differently in editor and console)
In editor: Move cursor up/down.
In console: Review previously executed commands.


#ALT+"-"
In script files and in console: Insert assignment operator " <- ".
```

d.

```r
# assign your year of birth to a variable "year"
year <- 1968

# make a vector "date" = (year, month, day) with your date of birth.
date <- c(1968, 9, 21)

# make a variable "a" that contains the value (2 + 3)*(10 - 3)^2.
a <- (2 + 3)*(10 - 3)^2

#assign the value 10 to variable "b" and let z be the sum of a and b.
b <- 10
z <- a + b

#assign the value 1 to "a", 2 to "b", 3 to "c" and 1 to "d".
a <- 1
b <- 2
c <- 3
d <- 1

#test whether "a" is equal to "b".
a == b
```

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

```r
#test whether "a"  is not equal to "c".
a != c
```

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

```r
#test whether "a" PLUS "b" is equal to "c" OR equal to "d".
#(here it is safest to use parentheses, and by the way, I think there is
#an error in the video here. I.e we can not write
# (a+b) == c | d as is suggested there.

((a+b) == c) | ((a+b) == d)
```

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

```r
#assign the numbers 2,3,4 and 5 to "e". (i.e. make a vector)
e <- c(2, 3, 4, 5)

#assign the numbers 12,13,14 and 15 to "f".
f <- c(12,13,14,15)

#test whether "a" is element in "e".
a %in% e
```

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

```r
#test whether "a" is element in "f".
a %in% f
```

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

```r
#make a vector y that contains the product by b of all elements in f.
#(so, y should be (24, 26, ...), but how to do it most easily with R
#given that you already defined b and f as variables?)
y <- b*f

#show that "f" MINUS 10 is "e". Use the logical comparison "==" in R.
(f - 10) == e
```

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

```r
#run ls() in the console to see your workspace variables. See the #environment #window
ls()
```

```
##  [1] "a"    "b"    "c"    "d"    "date" "e"    "f"    "y"    "year" "z"
```

```r
#run rm(list = ls()) to clear the workspace. Now run ls() again. You can
#run the whole R script to recreate the workspace variables. (CTRL+SHIFT+ENTER)
rm(list=ls())

#Write ?sum at the console. What happens?
#?sum
#  we get the internal help info about the "sum" function


#Create a vector z with 10-15 arbitrary numbers between 0 and 10.
z <- c(5, 4, 5, 3, 7, 5, 7, 8, 0, 10, 1, 3)


#write hist(z). What happens?
hist(z)
```


```
#In the plots-tab click "zoom".  What happens?
# -> the figure is magnified
```

3.3

```
#make a vector z of 5-10 arbitary numbers.

z <- c(5, 67, 33, 43, 41, 55)   # or try z <- sample(1:100, 10)

#Find the sum of elements in z
sum(z)
```

```
## [1] 244
```

```
#Find the average value of z
mean(z)
```

```
## [1] 40.66667
```

```
#Find the median value of z
median(z)
```

```
## [1] 42
```

```
#find the mean value of z^2
mean(z^2)
```

```
## [1] 2026.333
```

```
#make a vector z2 which starts with z but has NA as an
#additional element. (hint, c(x, y) will put vector y
# after vector x in a new vector)
z2 <- c(z, NA)

#try mean(z2)
mean(z2)
```

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

```
#try mean(z2, na.rm = TRUE)
mean(z2, na.rm = T)
```

```
## [1] 40.66667
```

```
#In the script editor, write "M <-  med". What happens? Hit TAB. What happens?

  #We get a list of options starting with "med". TAB selects one option.

#At the console, write "fact", and use this to find "factorial(10)"
factorial(10)
```

```
## [1] 3628800
```

```
#which is 10*9*8*...3*2*1. Find approximately factorial(100).
factorial(100)
```

```
## [1] 9.332622e+157
```

100! is a number starting 9 332 6 . . . . which has 157 digits.

3.4

a.

```
# clear your workspace. (rm(...) see above if you forgot)
rm(list = ls())

#Use ":" to make a vector x = 4, 5, 6, 7, 8, 9, 10 and a
#vector y = 100, 99, 98, ...., 3, 2, 1, 0.

x <- 4:10
y <- 100:0
```

```r
# find the length of y.
length(y)
```

```
## [1] 101
```

```r
# Use "seq" to make a vector z = 1, 4, 7, 10 and a vector
# w = 10, 8, 6, 4, 2, 0
z <- seq(from=1, to=10, by=3)
w <- seq(from=10, to=0, by=-2)

# Use "rep" to make a vector x = 1, 2, 3, 1, 2, 3, 1, 2, 3
rep(c(1,2,3), 3)
```

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

```r
# Use c(...) to make a vector y that starts with z and ends with w
c(z, w)
```

```
##  [1]  1  4  7 10 10  8  6  4  2  0
```

```r
# Make a vector f1 with 5 uniformly distributed random numbers
# between 0 and 10.
f1 <- runif(5, min=0, max=10)

# Make a vector f2 with 5 uniformly distributed random numbers
# between 0 and 10. Is f1 == f2?
f2 <- runif(5, min=0, max=10)

f1==f2
```

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

```r
# How can we ensure that the randomly generated numbers are
# the same each time we run our code? And why can this be of
# importance? Make a small sequence of code that ensures
# reproducible random generation of 5 uniform numbers as above.
```

This can be important if we have some experiment or simulation that we want to reproduce exactly. Also, if our code includes random numbers, and produces an error, it can be very difficult to track the error if we can not reproduce exactly the random input. To ensure the same random sequence in repetitions, we use `set.seed(...)`

```r
#set seed with some arbitrary number.
set.seed(12121)
f1 <- runif(5, min=0, max=10)

set.seed(12121)
f2 <- runif(5, min=0, max=10)
```

```r
f1 == f2
```

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

Now `f1` and `f2` are identical.

b.

```r
# Define the vector a as follows
a <- c(3, 5, 7, 9, 1, 1)

# extract the third element of a
a[3]
```

```
## [1] 7
```

```r
# extract the vector of all elements of a except the third
a[-3]
```

```
## [1] 3 5 9 1 1
```

```r
# extract elements in position 2 and 4
a[c(2,4)]
```

```
## [1] 5 9
```

```r
# extract all elements of a that are greater than 3
a[(a>3)]
```

```
## [1] 5 7 9
```

```r
# try head(a, 3) and tail(a, 3). What do you get?
head(a, 3)
```

```
## [1] 3 5 7
```

```r
tail(a, 3)
```

```
## [1] 9 1 1
```

```r
# change the first element of a to value 1.
a[1] <- 1

# change the second and third element to 0
a[c(2,3)] <- 0

# Set a back as originally defined. Let b be the vector a backwards
# hint: Google
a <- c(3, 5, 7, 9, 1, 1)
  #google suggest to use rev(..)
b <- rev(a)
```

```r
# What are the vectors a + b, and a*b? Try and see.
a+b
```

```
## [1]  4  6 16 16  6  4
```

```r
a*b
```

```
## [1]  3  5 63 63  5  3
```

```r
# What is the vector a + 10? Try and see.
a+10
```

```
## [1] 13 15 17 19 11 11
```

```r
# Find the vectors s1, s2 with elements in a sorted in
# (1) increasing, (2) decreasing order.
sort(a)
```

```
## [1] 1 1 3 5 7 9
```

```r
sort(a, decreasing = TRUE)
```

```
## [1] 9 7 5 3 1 1
```

```r
# What do you get if you try to add x = c(1, 2, 3)
# and y = c(1, 2, 3, 4)?
x <- c(1, 2, 3)
y <- c(1, 2, 3, 4)

x+y
```

```
## Warning in x + y: longer object length is not a multiple of shorter object
## length
```

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

We get a warning. What happens is that R tries to "recycle" the short vector `x` and add sequentially to `y`. But 4 is not a multiple of 3, so it doesn't really match. We get a result, but also a warning.

3.5

```r
install.packages("ggplot2")
```

This should give some messages, perhaps install necessary other packages first.

```r
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.0.5
```

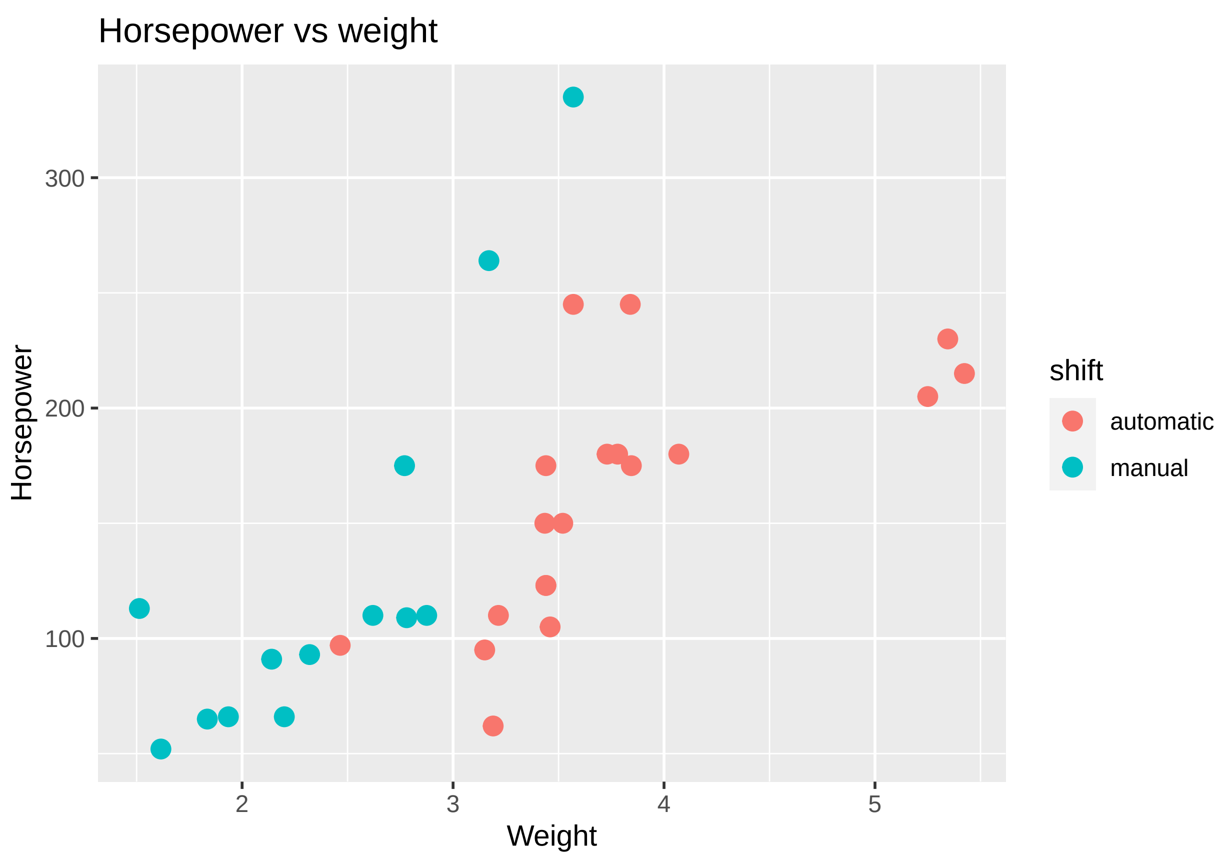
```r
data(mtcars)

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
```

```r
#Redefine variable "am" as a factor, just to make figure labels look right.
shift <- factor(mtcars$am, labels = c("automatic", "manual"))

#make a plot
ggplot(mtcars, aes(x = wt, y = hp, color = shift)) +
  geom_point(size = 3) +
  labs(title = "Horsepower vs weight",
       x = "Weight",
       y = "Horsepower")
```

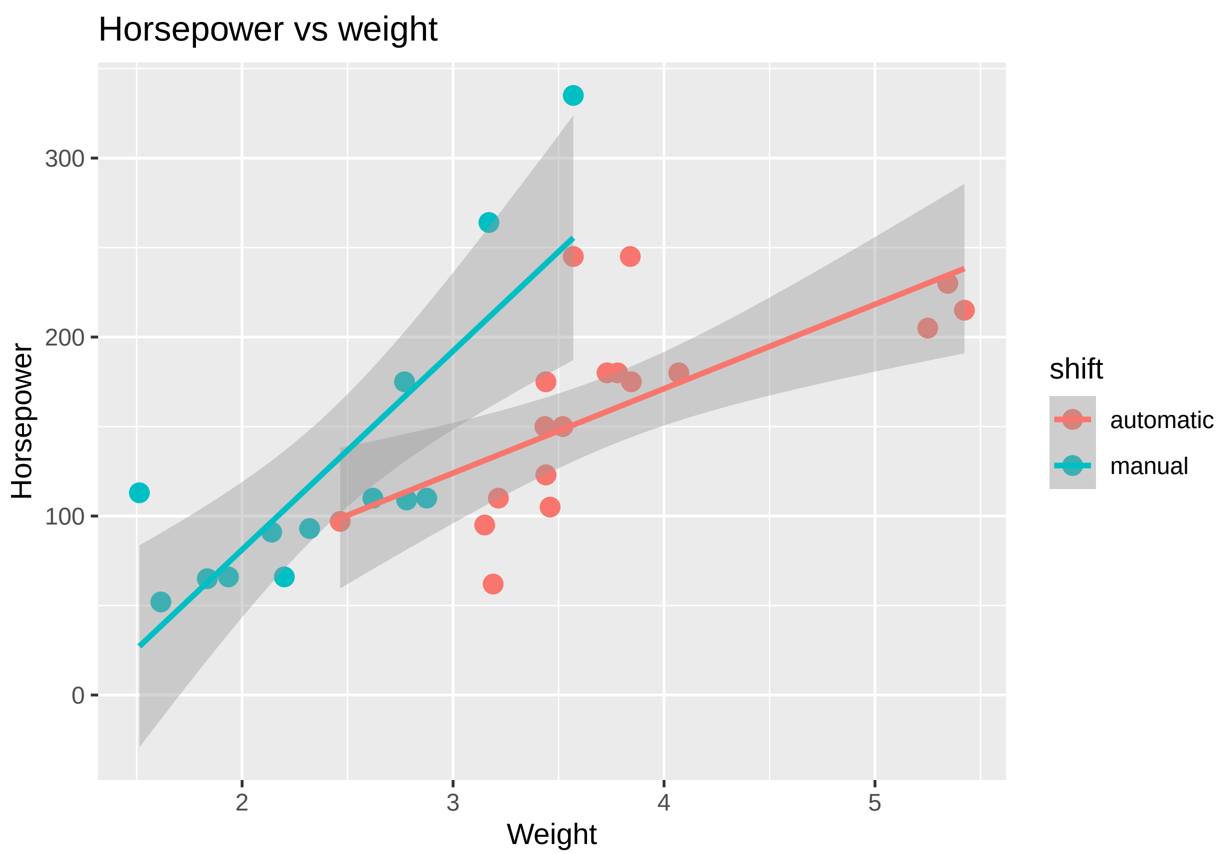


b.

We try to add the suggested code.

```
ggplot(mtcars, aes(x = wt, y = hp, color = shift)) +
  geom_point(size = 3) +
  labs(title = "Horsepower vs weight",
       x = "Weight",
       y = "Horsepower") +
   geom_smooth(method = lm)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



The extra code adds *estimated* regression lines to the two groups of data, and also indicates shaded areas where the *true* regression lines are likely to be found.

3.6

c. In my case, the code to read the file looks like below.

```
flights_NO <- read.csv("M:/Undervisning/Undervisningh21/Data/flights_NO.csv")
```

The file path as usual, will be different for each user. Note that the "automatic code" from "Import dataset" makes the name of the *dataframe* equal to the name of the *datafile*. This is in general *not necessary*. We may call the dataframe almost whatever we want. We use `head` to examine top rows of data:

```
head(flights_NO)
```

```
##   Origin.Code Destination.Code Dep.Time Arr.Time Flight Airline.Code
## 1         AES              BGO      715      800   4139           SK
## 2         AES              BGO      715      800   4139           SK
## 3         AES              BGO      715      800   4139           SK
## 4         AES              BGO      715      800   4139           SK
## 5         AES              BGO      715      800   4139           SK
## 6         AES              BGO     1100     1140   4147           SK
##        Alliance Origin.Country Destination.Country Day.of.Week Block.Mins Seats
## 1 Star Alliance         Norway              Norway      Monday         45   181
## 2 Star Alliance         Norway              Norway     Tuesday         45   141
## 3 Star Alliance         Norway              Norway   Wednesday         45   141
## 4 Star Alliance         Norway              Norway    Thursday         45   141
## 5 Star Alliance         Norway              Norway      Friday         45   141
## 6 Star Alliance         Norway              Norway      Monday         40    90
```

It appears that the data contains flight information.

d. We run the code

```
df2 <- subset(flights_NO, Day.of.Week == "Monday" & Seats > 50)
```

The code filters out to a dataframe `df2` all Monday fligths with more than 50 seats available in the plane. We find the number of such by

```
nrow(df2)
```

```
## [1] 488
```

and the total number in data is

```
nrow(flights_NO)
```

```
## [1] 4897
```

e. The suggested code is

```
#write df2 to disk with given file name.
write.csv(df2, "LargeMondayFlights.csv", row.names = FALSE)

#view the working directory, you should see the new file there:
dir()
```

```
##  [1] "Ch1Solutions.html"      "Ch1Solutions.pdf"       "Ch1Solutions.Rmd"
##  [4] "Ch2Solutions.html"      "Ch2Solutions.pdf"       "Ch2Solutions.Rmd"
##  [7] "Ch3Solutions.html"      "Ch3Solutions.pdf"       "Ch3Solutions.Rmd"
## [10] "Ch3Solutions_files"     "Ch3Solutions_full.html" "friendsfile.csv"
```

```
## [13] "LargeMondayFlights.csv"
```

And, yes - we see the new file there :-). I will remove it later, because I don't want it there.

f. Now we want to write to the main data folder. Again you need to figure out (or use autocomplete in R) the *path* to that folder. In fact the path is the same as you read from in part c. For me the code will be:

```
write.csv(df2, "M:/Undervisning/Undervisningh21/Data/LargeMondayFlights_NO.csv")
```

To see the content of that folder, I can write:

```
dir("M:/Undervisning/Undervisningh21/Data/")
```

```
##  [1] "AirBnBSing2.csv"            "alkfos.csv"
##  [3] "clock_auction.csv"          "Company_sales.csv"
##  [5] "Cruiseship.csv"             "Cruiseship4.csv"
##  [7] "desktop.ini"                "flat_prices.csv"
##  [9] "flights_NO.csv"             "Hospital_durations.csv"
## [11] "HotelAS.csv"                "LargeMondayFlights.csv"
## [13] "LargeMondayFlights_NO.csv" "log708data.7z"
## [15] "log708data.zip"             "meat_brands.csv"
## [17] "MetalAS.csv"                "Money_vs_time.csv"
## [19] "Mt.csv"                     "newdata.csv"
## [21] "Norfirms.csv"               "Nycflights2.csv"
## [23] "Tdur.csv"                   "TeleAS.csv"
## [25] "Trip_durations.csv"         "used_cars.csv"
## [27] "Wages.csv"                  "WaterWorld.csv"
## [29] "world95.csv"
```

We see all the files that were there already, plus the new one.

3.7

a. There was a minor "error" in this question. Using `?normal` or `help(normal)` does not work as intended. Turns out we can use `?Normal` with capital "N". Or, one can use `?distributions` and then find that the normal distributions can be found under `dnorm`.
   Then `?dnorm` will give the desired information. Here we can learn that there are four variants, among which `pnorm` gives cumulative probabilities. So then we can find the probabilities in the question as

```
pnorm(2.1)
```

```
## [1] 0.9821356
```

```r
(1 - pnorm(2.1))
```

```
## [1] 0.01786442
```

```r
# or:
pnorm(2.1, lower.tail = FALSE)
```

```
## [1] 0.01786442
```

```r
pnorm(12, mean = 10, sd = 3)
```

```
## [1] 0.7475075
```

```r
pnorm(10.9, mean = 10, sd = 3) - pnorm(8.4, mean = 10, sd = 3)
```

```
## [1] 0.32101
```

b. A Google search for "R calculate normal distribution" leads (among many other hits) to the web page at Tutorialspoint.

c. Reading carefully the internal or external information tells us we need to use the function `qnorm` to answer this question. So, to find b as specified in the question we need to do

```r
qnorm(0.80)
```

```
## [1] 0.8416212
```

We also note that the answer "looks right" as we are working on the standard normal distribution here. We could also check with the table in chapter 2.

d. This is similar to the question above, but using a general normal distribution. Obviously, the number c must satisfy $P[X \leq c] = 0.10$ (law of complement), so we can do

```r
qnorm(0.10, mean = 10, sd = 3)
```

```
## [1] 6.155345
```

Alternatively using the option `lower.tail = FALSE` we can write

```r
qnorm(0.90, mean = 10, sd = 3, lower.tail = FALSE)
```

```
## [1] 6.155345
```

e. From the documentation (and as shown in chapter 3) this is done via

```r
set.seed(3333)
my_x <- rnorm(30, mean = 10, sd = 3)

#compute mean and standard deviation
M <- mean(my_x)
S <- sd(my_x)
```

```r
#show result:
c(M, S)
```

```
## [1] 10.720201  2.497178
```

We see the numbers fairly close to the "true" parameter values 10 and 3.

3.8

a. - d. We could go as follows.

```r
a <- c("Aksel", "Amanda", "Hege", "Jakob", "Einar",
       "Geir", "Anette", "Ketil", "Lise", "Falko")
b <- c(6, 10, 49, 62, 60,
       53, 39, 54, 56, 43)
c <- c("Blond", "Blond", "Dark", "Dark", "Brown",
       "Gray", "Dark", "Gray", "Dark", "Dark")
d <- c(6, 10, 20, 51, 51,
       8, 10, 18, 10, 6)
```

e. - f. This can be done in two simple ways either (i) create the dataframe and rename columns or (ii) create dataframe and set names at once. I prefer (ii) as it is more compact code.

```r
# method (i)
friends <- data.frame(a, b, c, d)
names(friends) <- c("name", "age", "hair", "time")
head(friends)
```

```
##     name age  hair time
## 1  Aksel   6 Blond    6
## 2 Amanda  10 Blond   10
## 3   Hege  49  Dark   20
## 4  Jakob  62  Dark   51
## 5  Einar  60 Brown   51
## 6   Geir  53  Gray    8
```

```r
#method (ii)
friends <- data.frame(name = a, age = b, hair = c, time = d)
head(friends)
```

```
##     name age  hair time
## 1  Aksel   6 Blond    6
## 2 Amanda  10 Blond   10
## 3   Hege  49  Dark   20
## 4  Jakob  62  Dark   51
## 5  Einar  60 Brown   51
```

```
## 6   Geir  53  Gray    8
```

We see the same result.

g. - h.

```r
#get subset
friends2 <- subset(friends, age < 20)

#count rows
nrow(friends2)
```

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

i. Again using `subset`.

```r
friends3 <- subset(friends, (age > 20) & (time > 10))
head(friends3)
```

```
##    name age  hair time
## 3  Hege  49  Dark   20
## 4 Jakob  62  Dark   51
## 5 Einar  60 Brown   51
## 8 Ketil  54  Gray   18
```

NOTE: The parentheses in `(age > 20) & (time > 10)` are not strictly necessary, but it makes the code clearer and "safer" than writing `age > 20 & time > 10`.

j. - k. New code:

```r
#define new vector
not_seen <- c(0, 0, 0, 60, 50,
              4, 4, 7, 5, 6)
#add vector to data frame.
friends$not_seen <- not_seen
head(friends)
```

```
##     name age  hair time not_seen
## 1  Aksel   6 Blond    6        0
## 2 Amanda  10 Blond   10        0
## 3   Hege  49  Dark   20        0
## 4  Jakob  62  Dark   51       60
## 5  Einar  60 Brown   51       50
## 6   Geir  53  Gray    8        4
```

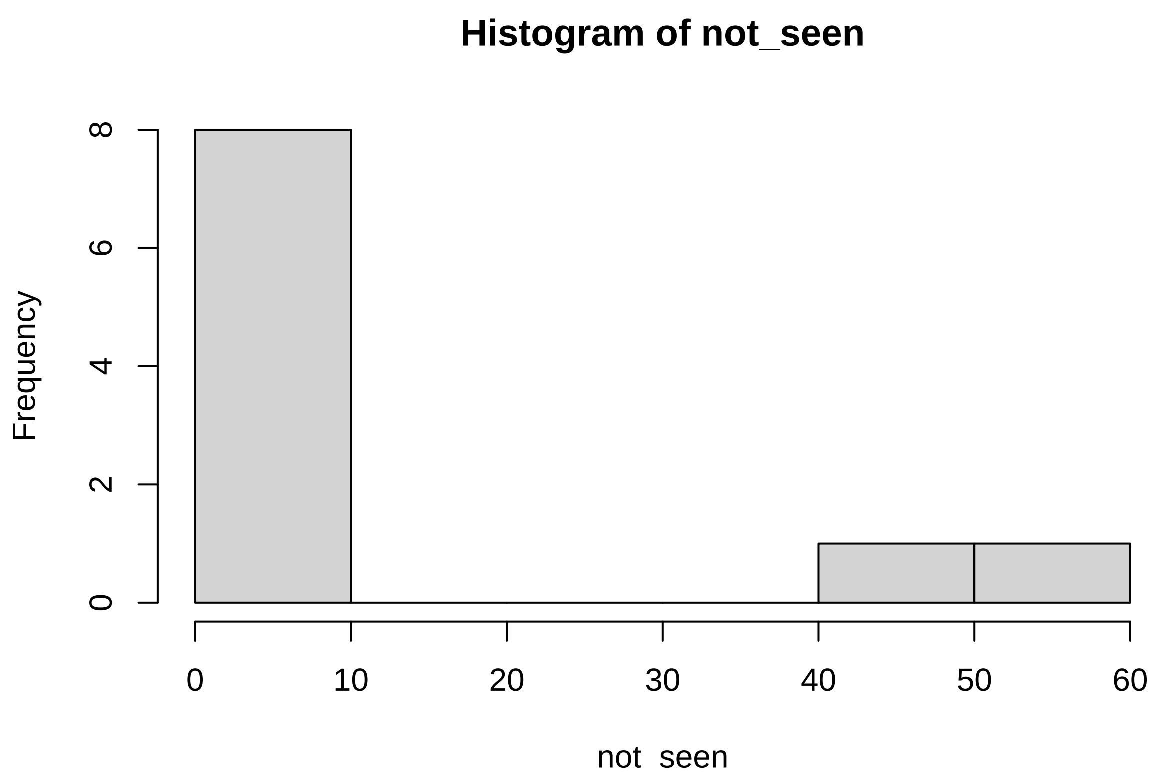
l. OK!

3.9

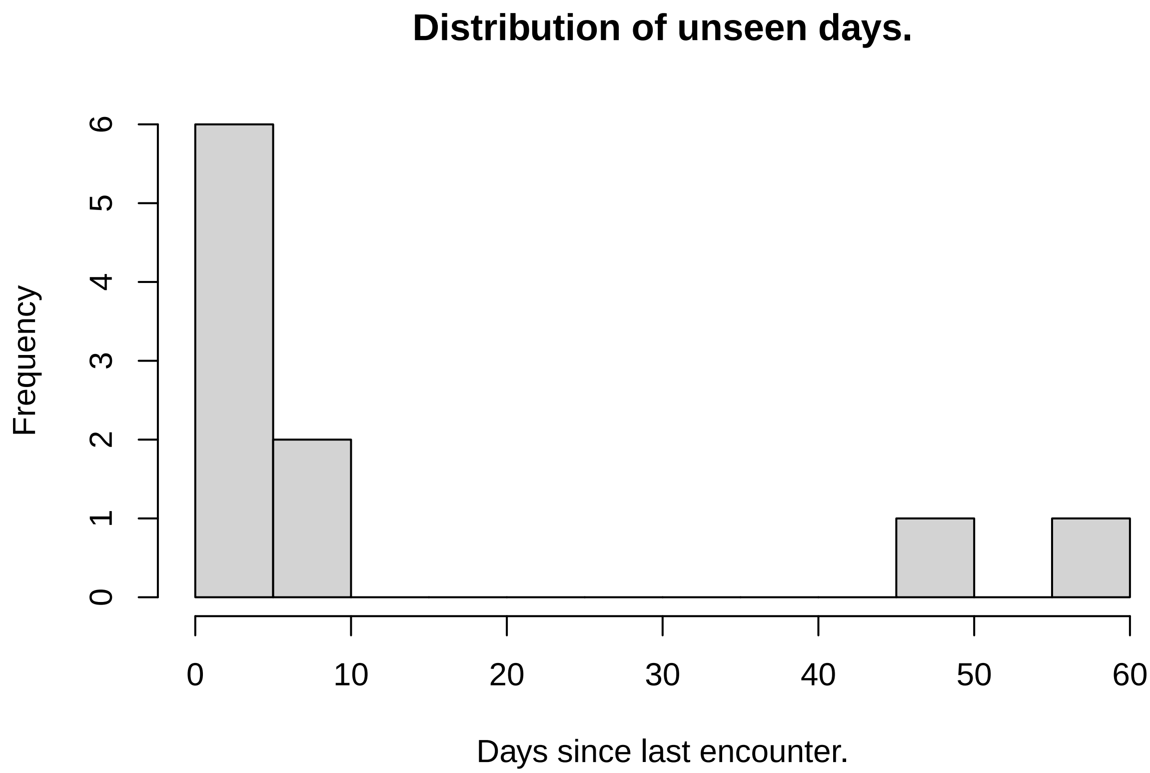
Now, continuing to work on the `friends` dataframe.

a. Writing e.g. `?hist` at the console opens help for this function. There are some parameters to play with. For example we can adjust the number of break points in the histogram as below.

```
with(friends, hist(not_seen))
```



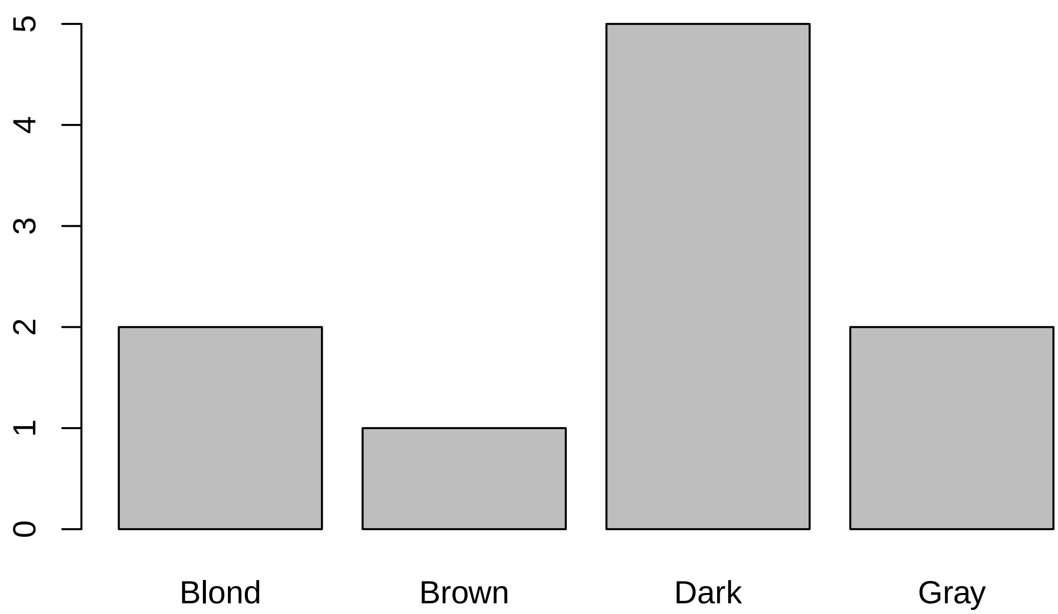

```
#or
with(friends, hist(not_seen,
                   breaks = 14,
                   main = "Distribution of unseen days.",
                   xlab = "Days since last encounter."))
```



The distribution is very skewed!

b. This was a little tricky? Going to Google, searching for "R barplot categorical data" led to many hits (as always). This link has a solution under "Categorical data". First we need to count the occurences, then call the `barplot` function.

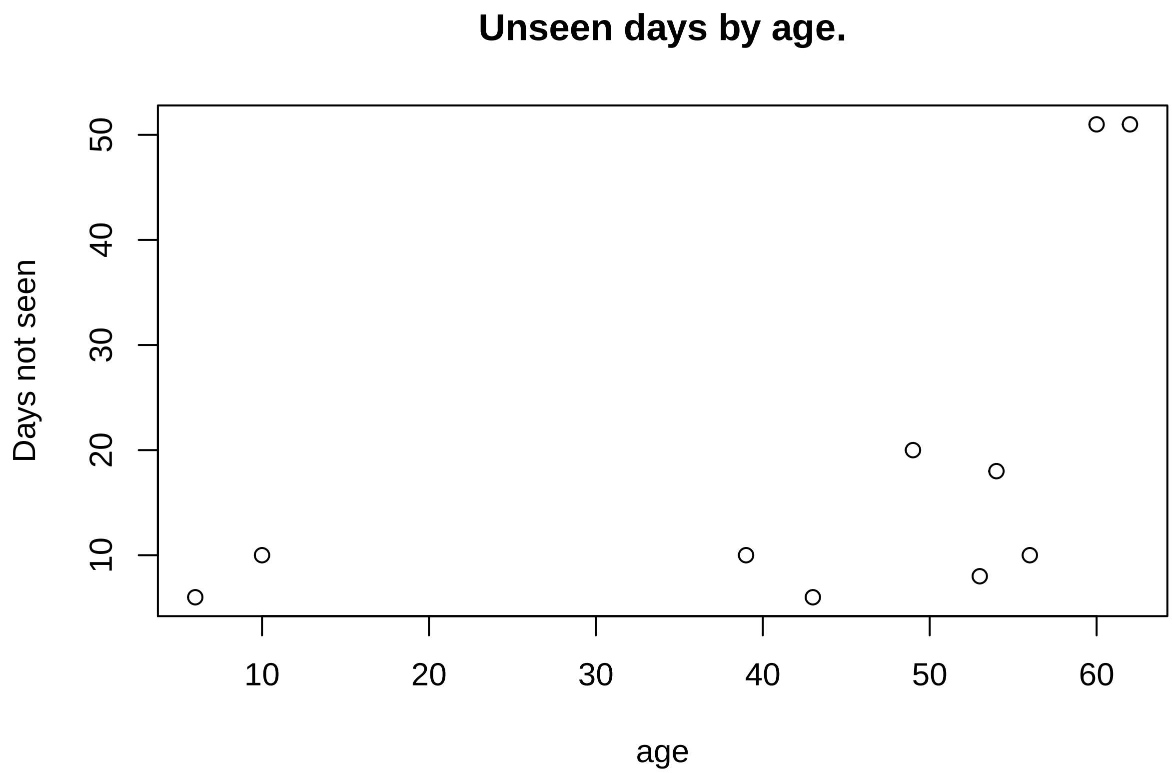
```
count <- table(friends$hair)
barplot(count)
```




```
#or we could do
#with(friends, barplot(table(hair)))
```

c. Here we can use the basic `plot` function.

```
with(friends, plot(age, time ,
                   main = "Unseen days by age.",
                   ylab = "Days not seen"))
```



NOTE: In the code we can choose to use `with(friends, ...)` or use the `$` method to pull out vectors from `friends` so the plot above could be created by

```
plot(friends$age, friends$time,
          main = "Unseen days by age.",
          ylab = "Days not seen",
          xlab = "age")
```

d. We write the data frame to our working directory as follows.

```
write.csv(friends, "friendsfile.csv", row.names = FALSE)
```

Note: Without the `row.names = FALSE` setting, R will add a column with numbers to the file. This column is included if you re-read the file into a new dataframe. Not a big problem, but usually unneccessary.

e. Read the file back. Printing first rows shows we get the same data back.

```
friends2 <- read.csv("friendsfile.csv")
head(friends)
```

```
##      name age  hair time not_seen
## 1   Aksel   6 Blond    6        0
## 2 Amanda  10 Blond   10        0
```

```
## 3    Hege  49  Dark   20        0
## 4   Jakob  62  Dark   51       60
## 5   Einar  60 Brown   51       50
## 6    Geir  53  Gray    8        4
```

```r
head(friends2)
```

```
##      name age  hair time not_seen
## 1   Aksel   6 Blond    6        0
## 2  Amanda  10 Blond   10        0
## 3    Hege  49  Dark   20        0
## 4   Jakob  62  Dark   51       60
## 5   Einar  60 Brown   51       50
## 6    Geir  53  Gray    8        4
```

NOTE: A careful check in the environment shows that the data types are changed slightly when re-reading the data from file. Some variables in `friends` that were `numeric` are interpreted as `integer` in `friends2`. This will usually not cause problems, if necessary we can use functions like `as.numeric` to force data back to same type. Also, modern data-analysis packages like `readr`, `dplyr` includes functions to test carefully that dataframes are identical, as well as read-and-write functions with more control on data types.

3.10

a. - b. In my folder system I can find the file as

```r
w95 <- read.csv("M:/Undervisning/Undervisningh21/Data/world95.csv")
names(w95)
```

```
##  [1] "country"       "population"    "density"       "urban"
##  [5] "religion"      "lifeexpf"      "lifeexpm"      "literacy"
##  [9] "pop_incr"      "babymort"      "gdp_cap"       "region"
## [13] "calories"      "aids"          "birth_rt"      "death_rt"
## [17] "aids_rt"       "log_gdp"       "lg_aidsr"      "b_to_d"
## [21] "fertilty"      "log_pop"       "cropgrow"      "lit_male"
## [25] "lit_fema"      "climate"       "lit_ratio"     "main_religion"
## [29] "inv_gdp"
```

Lots of demographic variables.

c. This will give number of columns and rows

```r
#write out in one row:
c(ncol(w95), nrow(w95))
```

```
## [1]  29 109
```

d.

```r
head(w95, 10)
```

```
##         country population density urban religion lifeexpf lifeexpm literacy
## 1      Austria       8000    94.0    58 Catholic       79       73       99
## 2      Belgium      10100   329.0    96 Catholic       79       73       99
## 3       Canada      29100     2.8    77 Catholic       81       74       97
## 4       France      58000   105.0    73 Catholic       82       74       99
## 5      Ireland       3600    51.0    57 Catholic       78       73       98
## 6        Italy      58100   188.0    69 Catholic       81       74       97
## 7  Netherlands      15400   366.0    89 Catholic       81       75       99
## 8     Portugal      10500   108.0    34 Catholic       78       71       85
## 9        Spain      39200    77.0    78 Catholic       81       74       95
## 10 Switzerland       7000   170.0    62 Catholic       82       75       99
##    pop_incr babymort gdp_cap region calories  aids birth_rt death_rt  aids_rt
## 1      0.20      6.7   18396   OECD     3495  1150       12     11.0 14.37500
## 2      0.20      7.2   17912   OECD       NA  1603       12     11.0 15.87129
## 3      0.70      6.8   19904   OECD     3482  9511       14      8.0 32.68385
## 4      0.47      6.7   18944   OECD     3465 30003       13      9.3 51.72931
## 5      0.30      7.4   12170   OECD     3778   392       14      9.0 10.88889
## 6      0.21      7.6   17500   OECD     3504 21770       11     10.0 38.05944
## 7      0.58      6.3   17245   OECD     3151  3055       13      9.0 19.83766
## 8      0.36      9.2    9000   OECD       NA  1811       12     10.0 18.29293
## 9      0.25      6.9   13047   OECD     3572 24202       11      9.0 61.73980
## 10     0.70      6.2   22384   OECD     3562  3662       12      9.0 52.31429
##     log_gdp lg_aidsr   b_to_d fertilty  log_pop cropgrow lit_male lit_fema
## 1  4.264723 1.704204 1.090909     1.50 3.903090       17       NA       NA
## 2  4.253144 1.738291 1.090909     1.70 4.004321       24       NA       NA
## 3  4.298940 2.008476 1.750000     1.80 4.463893        5       NA       NA
## 4  4.277472 2.201645 1.397849     1.80 4.763428       32       NA       NA
## 5  4.085291 1.612118 1.555556     1.99 3.556303       14       NA       NA
## 6  4.243038 2.070582 1.100000     1.30 4.764176       32       98       96
## 7  4.236663 1.817599 1.444444     1.58 4.187521       26       NA       NA
## 8  3.954243 1.788367 1.200000     1.50 4.021189       32       89       82
## 9  4.115511 2.280936 1.222222     1.40 4.593286       31       97       93
## 10 4.349938 2.206602 1.333333     1.60 3.845098       10       NA       NA
##          climate lit_ratio main_religion      inv_gdp
## 1      temperate        NA      Catholic 5.435964e-05
## 2      temperate        NA      Catholic 5.582849e-05
## 3  arctic / temp        NA      Catholic 5.024116e-05
## 4      temperate        NA      Catholic 5.278716e-05
## 5      temperate        NA      Catholic 8.216927e-05
## 6  mediterranean  1.020833      Catholic 5.714286e-05
## 7      temperate        NA      Catholic 5.798782e-05
## 8       maritime  1.085366      Catholic 1.111111e-04
```

```
## 9      temperate  1.043011      Catholic 7.664597e-05
## 10     temperate        NA      Catholic 4.467477e-05
```

e. We get a spreadsheet-type look at data.

f. We can do

```
malemean <- mean(w95$lifeexpm)
femalemean <- mean(w95$lifeexpf)

#make a vector
means <- c(malemean, femalemean)
#name the components
names(means) <- c("M", "F")
#print means:
means
```

```
##        M        F
## 64.91743 70.15596
```

Males were at about 65 years, females at 70 on average.

g. We can just copy the code above and make minor changes.

```
malemed <- median(w95$lifeexpm)
femalemed <- median(w95$lifeexpf)

#make a vector
medians <- c(malemed, femalemed)
#name the components
names(medians)<- c("M", "F")
#print medians:
medians
```

```
##  M  F
## 67 74
```

The medians are somewhat larger than the means, so probably a few countries are particularly low, drawing the means down. I.e. there is some skewness. The way to see this is of course to visualize, which is next.

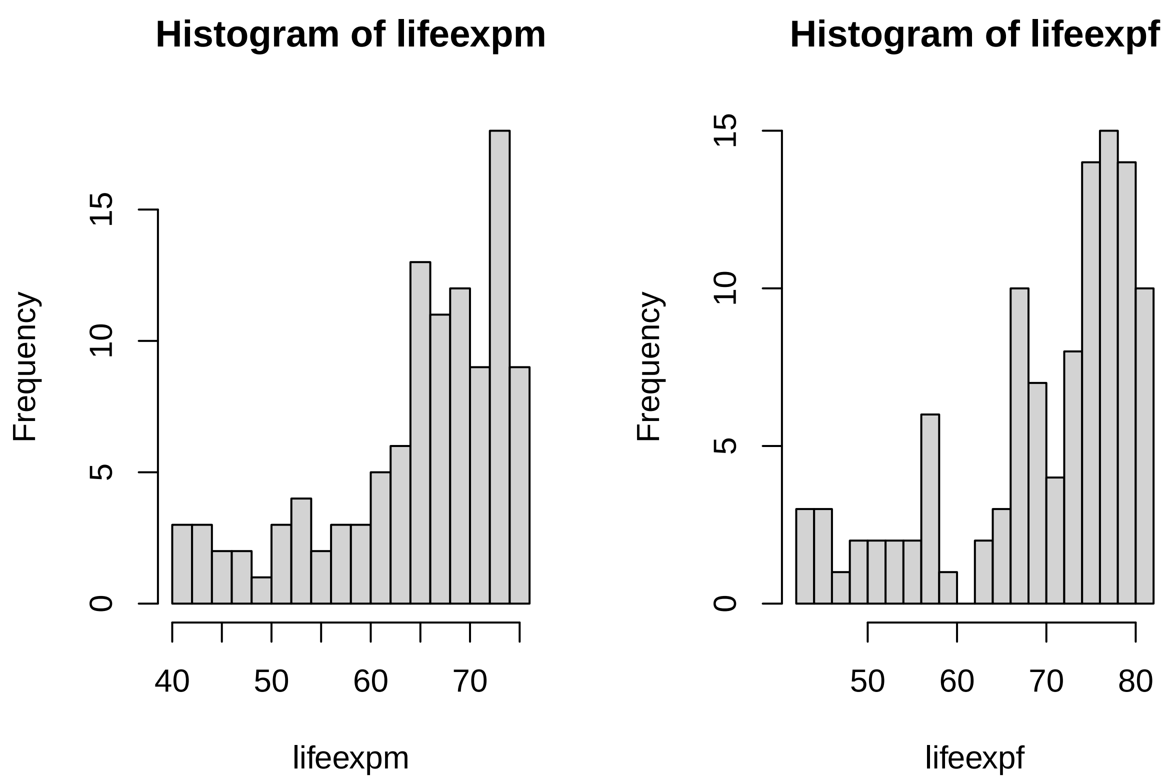
h.

```
#allow side-by-side plots:
par(mfrow = c(1,2))

with(w95, hist(lifeexpm,
               breaks = 15))

with(w95, hist(lifeexpf,
```

```
                  breaks = 15))
```




```
#reset plot parameters:
par(mfrow = c(1,1))
```

These histograms show the skewness to the left in both variables. i. Ok, so we go

```
MM <- max(w95$lifeexpm)
MM
```

```
## [1] 76
```

Ok, 76 years was the maximum.

j. Which countries reached max?

```
#make a subset: Note, we use the variable name MM (not the number #76) to make the cod
max_df <- subset(w95, lifeexpm == MM)

#list the names (in the variable "country")
max_df$country
```

```
## [1] "Iceland"    "Japan"      "Israel"     "Costa Rica"
```

```r
# or to see more related data:
head(max_df)
```

```
##       country population density urban religion lifeexpf lifeexpm literacy
## 16    Iceland        263     2.5    91 Protstnt       81       76      100
## 38      Japan     125500   330.0    77 Buddhist       82       76       99
## 72     Israel       5400   238.0    92   Jewish       80       76       92
## 94 Costa Rica       3300    64.0    47 Catholic       79       76       93
##    pop_incr babymort gdp_cap        region calories aids birth_rt death_rt
## 16     1.10      4.0   17241          OECD       NA   31       16        7
## 38     0.30      4.4   19860  Pacific/Asia     2956  713       11        7
## 72     2.22      8.6   13066   Middle East       NA  279       21        7
## 94     2.30     11.0    2031 Latin America     2808  587       26        4
##       aids_rt  log_gdp  lg_aidsr   b_to_d fertilty  log_pop cropgrow lit_male
## 16 10.3333333 4.236562 1.5953210 2.285714     2.11 2.419956        1       NA
## 38  0.5681275 4.297979 0.8930775 1.571429     1.55 5.098644       13       NA
## 72  5.1666667 4.116143 1.3888076 3.000000     2.83 3.732394       17       95
## 94 17.7878788 3.307710 1.7783811 6.500000     3.10 3.518514        6       93
##    lit_fema       climate lit_ratio main_religion      inv_gdp
## 16       NA     temperate        NA      Protstnt 5.800128e-05
## 38       NA mediterranean        NA      Buddhist 5.035247e-05
## 72       89     temperate  1.067416         Other 7.653452e-05
## 94       93      tropical  1.000000      Catholic 4.923683e-04
```

k.

```r
meanpop <- mean(w95$population)
medianpop <- median(w95$population)

c(meanpop, medianpop)
```

```
## [1] 47723.88 10400.00
```

The mean is about 47 million, while the median is about 10.4 million. A striking difference. What proportion of countries have less than the mean population? We could subset the dataframe and count rows.

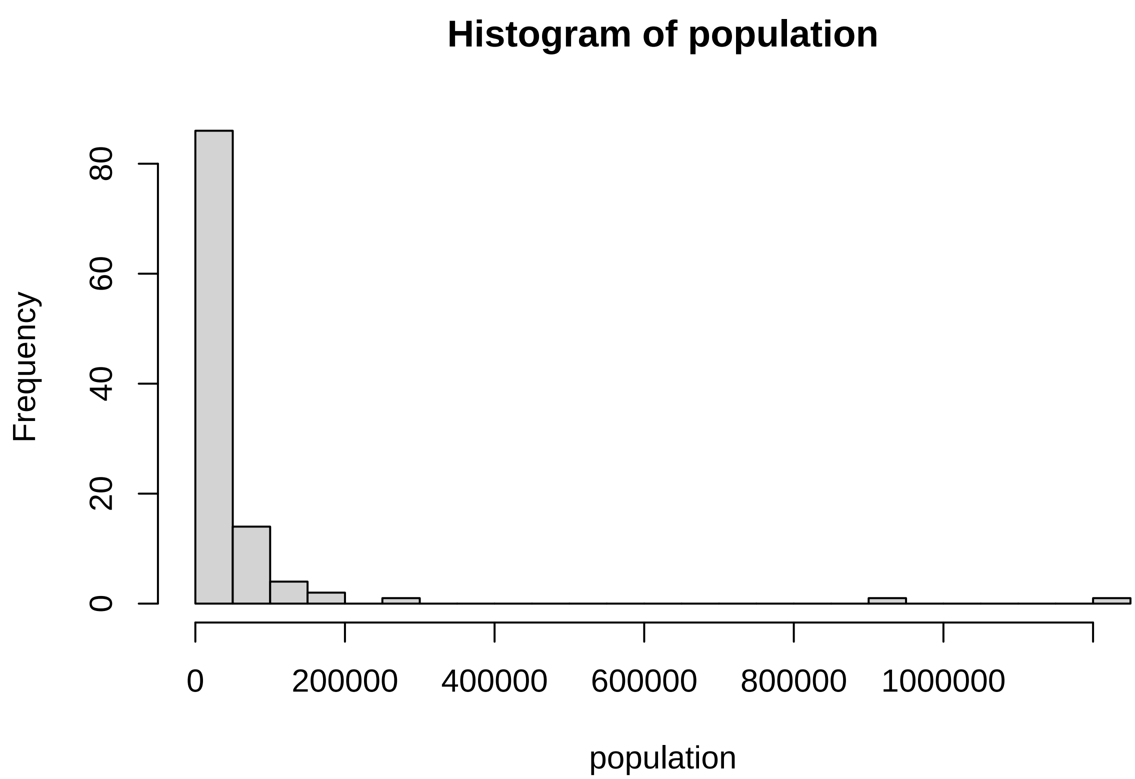
```r
LessThanMean <- subset(w95, population < meanpop)
#find the count:
count <- nrow(LessThanMean)
#find the proportion
proportion <- count/nrow(w95)

c(count, proportion)
```

```
## [1] 86.0000000  0.7889908
```

So, 86 countries out of 109 were below the mean population. That's about 80% of the countries. That shows in this case the mean is not a "typical value" for population sizes. The median of course by definition splits the sample 50/50. Histogram follows.

```r
with(w95, hist(population, breaks = 30))
```



We see two countries in particular sticking out, that would be China and India. This explains much of the problems with the mean in this case.

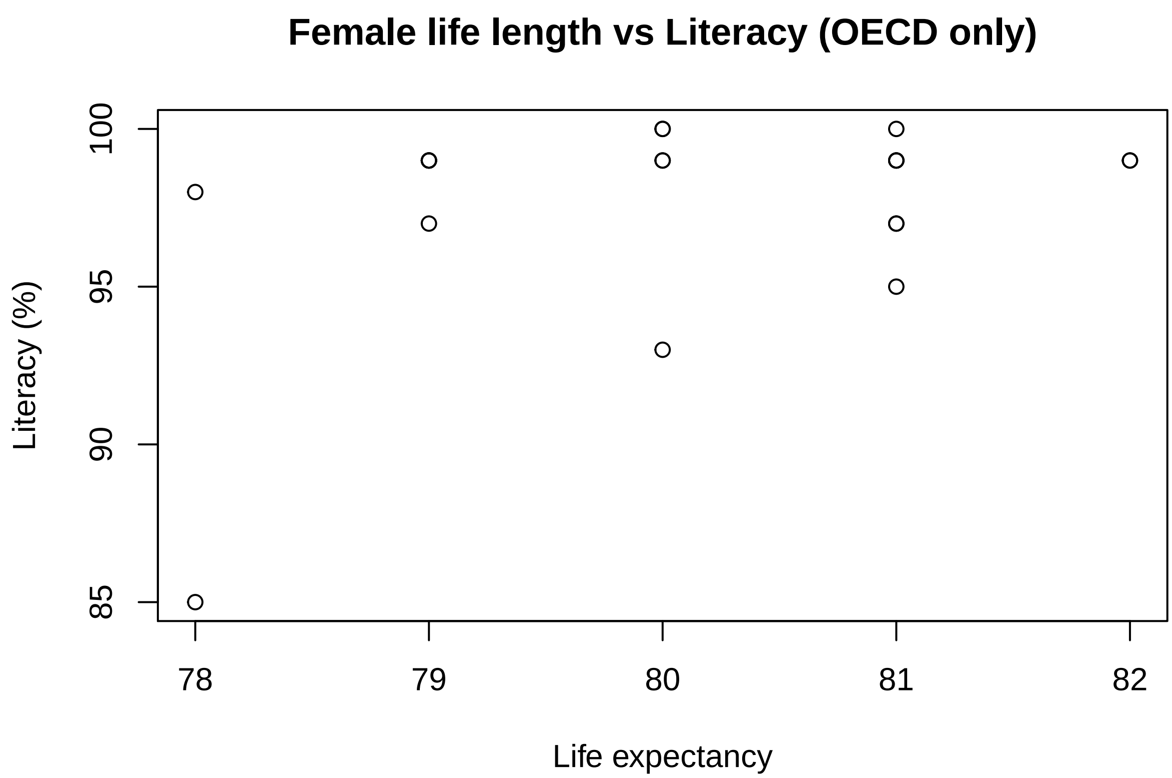
l. We use the `plot` function.

```r
with(w95, plot(lifeexpf, literacy,
               main = "Female life length vs Literacy",
               xlab = "Life expectancy",
               ylab = "Literacy (%)"))
```

We see that the two variables have a strong correlation, and again, we can suspect that both are related to some of the same underlying variables.

m. We just subset and plot as suggested.

```
OECD95 <- subset(w95, region == "OECD")
with(OECD95, plot(lifeexpf, literacy,
               main = "Female life length vs Literacy (OECD only)",
               xlab = "Life expectancy",
               ylab = "Literacy (%)"))
```



n.

```
Regions <- table(w95$region)
Regions
```

```
##
##        Africa    East Europe Latin America   Middle East          OECD
##            19             14            21            17            21
##  Pacific/Asia
##            17
```

```
#OR: with(w95, table(region))
```

o.

```
mean(w95$calories)
```

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

```
mean(w95$calories, na.rm = T)
```

```
## [1] 2753.827
```

We get `NA` because there are some `NA` values in the original variable.

p. Let's try

```
table(is.na(w95$calories))
```

```
##
## FALSE  TRUE
##    75    34
```

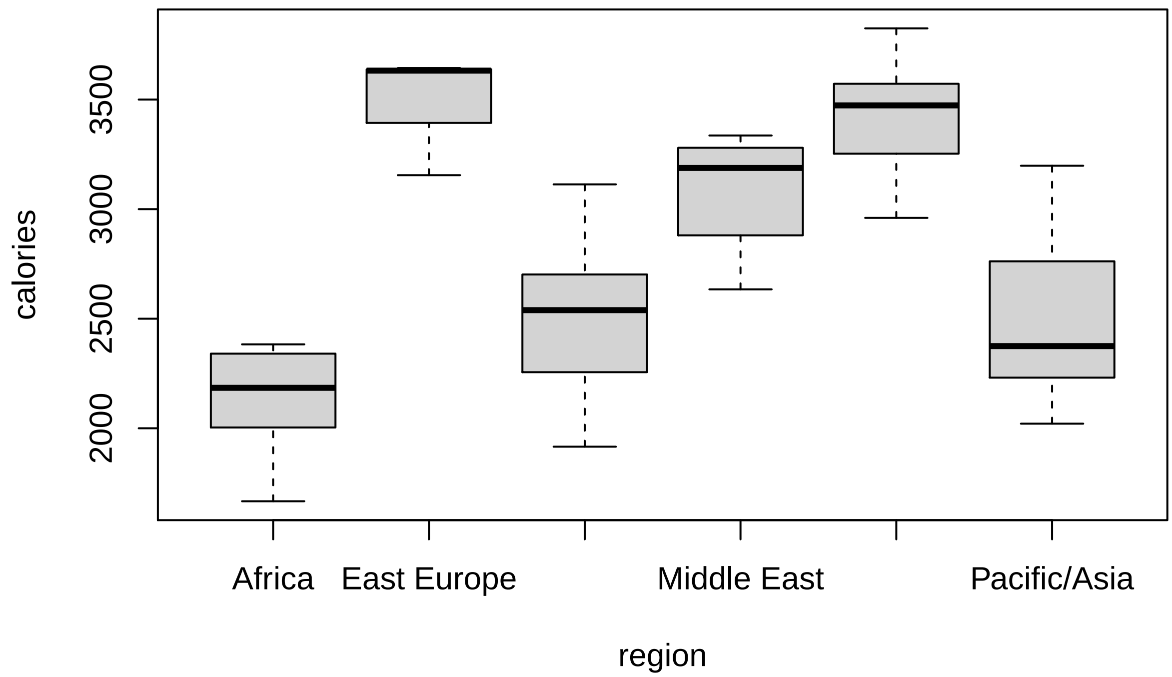
There are in fact 34 countries with no value for `calories`. This makes using the calculated mean value a bit dangerous. We can do the following (not asked for) code to see what regions have most NA's. In particular East Europe and the Middle East have high proportions of NA.

```
table(w95$region, is.na(w95$calories))
```

```
##
##                  FALSE TRUE
##   Africa            16    3
##   East Europe        3   11
##   Latin America     19    2
##   Middle East        8    9
##   OECD              18    3
##   Pacific/Asia      11    6
```

q. Ok

```
with(w95, boxplot(calories ~ region))
```



Ok, some clear differences are observed. With the above information about NA, we should be careful about some of the regional data.