ST523/ST813 Statistical Modelling

Introductory Self Study Tutorial
and Introduction to R

1 Short introduction

In this lab we will start with a general review of R, how to create functions, how to read in data, how to visualise data and a very basic starting analysis of the "Hearing Threshold" dataset, which is available on itslearning.

This dataset corresponds to a study from Masterson et. al. (Prevalence of Hearing Loss in the United States by Industry, 2013) about the analysis of hearing loss of workers in U.S industries. We consider an anonymized subset of the original data. A description of the variables is found in Table 1.

2 R generals

R is a system for statistical analyses and graphics. R is both a software and a programming language freely distributed under the terms of the GNU General Public Licence; its development and distribution are carried out by several statisticians known as the R Core Team. You can find more information at `http://cran.r-project.org`.

3 Installation

3.1 Installing R

You can find precompiled versions of R directly in the following mirror sites:
`http://cran.r-project.org`

Click on the link to download R for the operating system (Windows, Mac, or Linux) and follow the indicated steps. Eventually, open the downloaded file after it has finished downloading and follow the instructions.

Table 1: Variables in `HearingThreshold.csv`. Hearing thresholds are measured in dB at different frequencies. Hearing impairment begins at ca. 25 dB.

`variable`	explanation
`id`	person id
`test_date`	date of hearing test
`r500`	Right ear hearing threshold at 500 dB
`r1k`	Right ear hearing threshold at 1k (=1000 dB)
`r2k`	Right ear hearing threshold at 2k
`r3k`	Right ear hearing threshold at 3k
`r4k`	Right ear hearing threshold at 4k
`r6k`	Right ear hearing threshold at 6k
`r8k`	Right ear hearing threshold at 8k
`l500`	Left ear hearing threshold at 500
`l1k`	Left ear hearing threshold at 1k
`l2k`	Left ear hearing threshold at 2k
`l3k`	Left ear hearing threshold at 3k
`l4k`	Left ear hearing threshold at 4k
`l6k`	Left ear hearing threshold at 6k
`l8k`	Left ear hearing threshold at 8k
`region`	U.S. geographical region: MA = Mid-Atlantic; MW = Midwest; NE = New England; SO = South; SW = Southwest; WE = West
`gender`	M=male F=female
`age`	age in years
`age_group`	grouped age: 1=18-25 2=26-35 3=36-45 4=46-55 5=56-65
`industry`	Industry type according to NAICS
`naics`	North-American Industry Classification System Code

3.2 Installing RStudio

Although you can use the R graphical user interface for code editing, you are strongly encouraged to install the editor RStudio, which will make your life much simpler when working with R. To download and install this editor, go to `http://www.rstudio.com`, click on download, click on the file after it has finished downloading, and follow the installation instructions.

4 R material

Check out the link `https://r4ds.hadley.nz/` for a useful online book about R.

Note further that `https://www.rstudio.com/resources/cheatsheets/` contains a collection of cheatsheets for the use of R. Many are highlevel but you might check the base R cheatsheet for basic functions (if you are rather new to R) or the ggplot cheatsheet for use of the ggplot package which gives access to improved plotting functions.

5 Getting started

At the beginning, make sure to point R to your working directory. Now, let's say that you want to work on a folder "C:/Documents/ST813/Exercises/", type the following code to set your working directory:

```
setwd("C:/Documents/ST813/Exercises/")
```

Alternatively, you may use the Files pane in the lower right corner of the RStudio window to find a directory and set it as working directory, either from the menu (Session → Set Working Directory → Files Pane Location) or directly in the Files pane (choose More → Set As Working Directory).

Then read the "HearingThresholds.csv" file so we can run a few plots on the data. To do that, type the following code:

```
data <- read.csv("HearingThresholds.csv", header = TRUE)
```

To make sure that R read correctly the files, you can ask R to show you the first few rows of the tables by typing

```
data[1:3,]

##        id  test_date l500 l1k l2k l3k l4k l6k l8k r500 r1k
## 1 4753714 2004-12-21   25  20  16  10  15  20  11   26  24
## 2 5288679 2006-01-17   21  17  15  16  31  21  36    8   1
## 3 2485439 2000-07-19   15  18  25  10  24  44  60   11   6
##   r2k r3k r4k r6k r8k region gender age_group age
## 1  11  12  19  28  11     WE      F     56-65  56
## 2   5   7  33  51  46             M     46-55  52
## 3  14  16  13  41  37     WE      M     46-55  47
##                                  industry  naics
## 1 Scheduled Passenger Air Transportation 481111
## 2 Couriers and Express Delivery Services 492110
## 3     Other Aluminum Rolling and Drawing 331319
```

Alternatively, you can use function **`head`** to view the first six rows as:

```
head(data)
```

It is useful to know the **`class`** of the object data by typing

```
class(data)

## [1] "data.frame"
```

The variable **`data`** is internally stored as a **data frame** object, i.e. R's built-in format to store datasets. We can refer to each column, say the **`gender`** column, as **`data$gender`**, this is by concatenating the name of the data frame and the name of the column using the dollar sign "$". Data frames allow to have in the same table columns with numerical values and columns with character values.

New variables can be generated similarly, e.g. the below code generates a variable **`avg1k`** (as part of the data frame) the contains the hearing threshold average over both ears (left and right) at 1kHz:

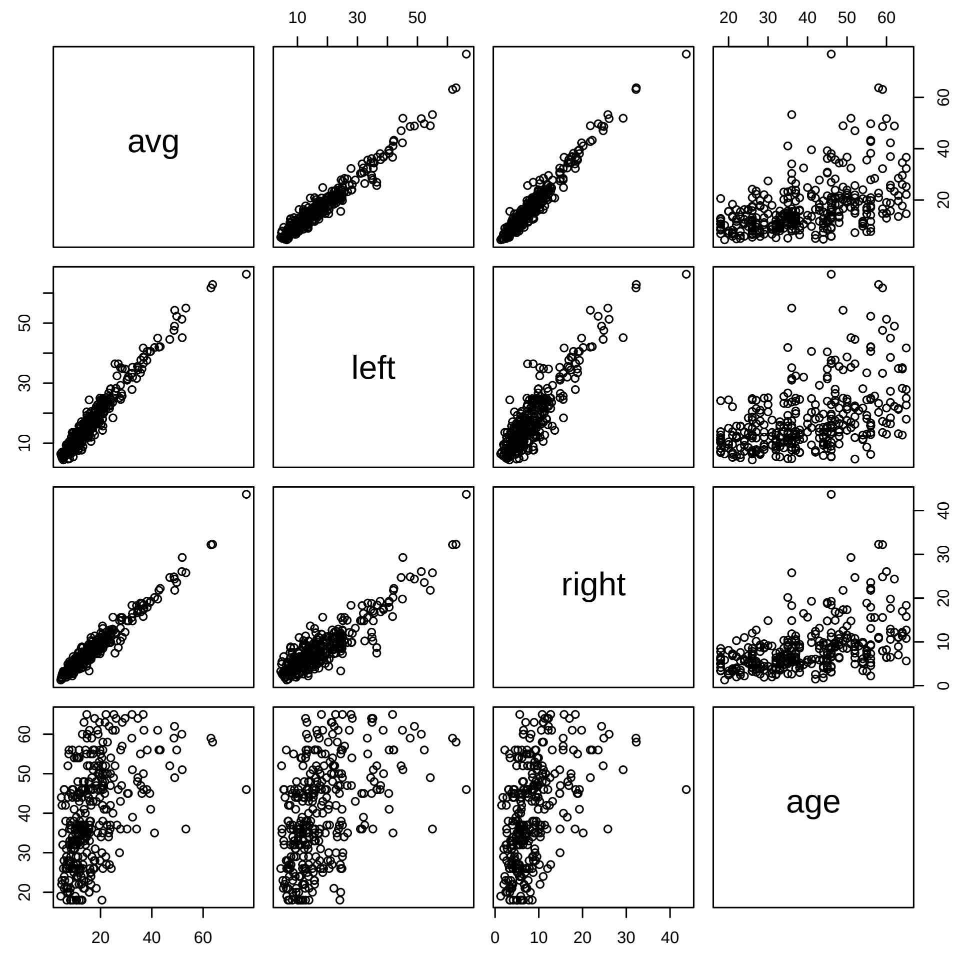
```
data$avg1k=(data$l1k+data$r1k)/2
```

Please generate as a next step a variable **`data$avg`** containing for each subject the average hearing threshold over all 14 frequencies from both ears, as well as average thresholds **`data$left`** and **`data$right`** over all frequencies from the left and right ear separately.

6 Plotting functions

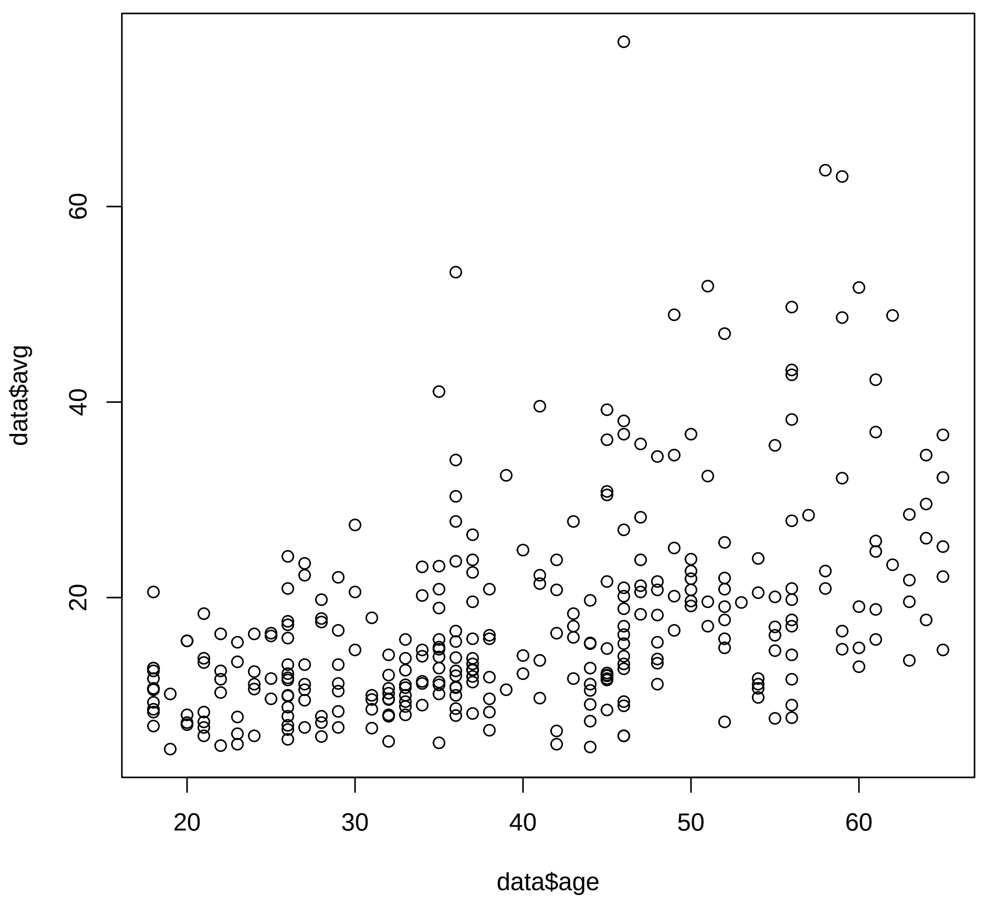
To take a first look at the data, let's produce a **scatter plot** of the average hearing thresholds `data$avg`, `data$left` and `data$right` against each other and against age. Before plotting, we are going to select the variables we are interested in from the data frame.

```
data_plot=subset(data, select=c(avg, left, right, age))
plot(data_plot)
```



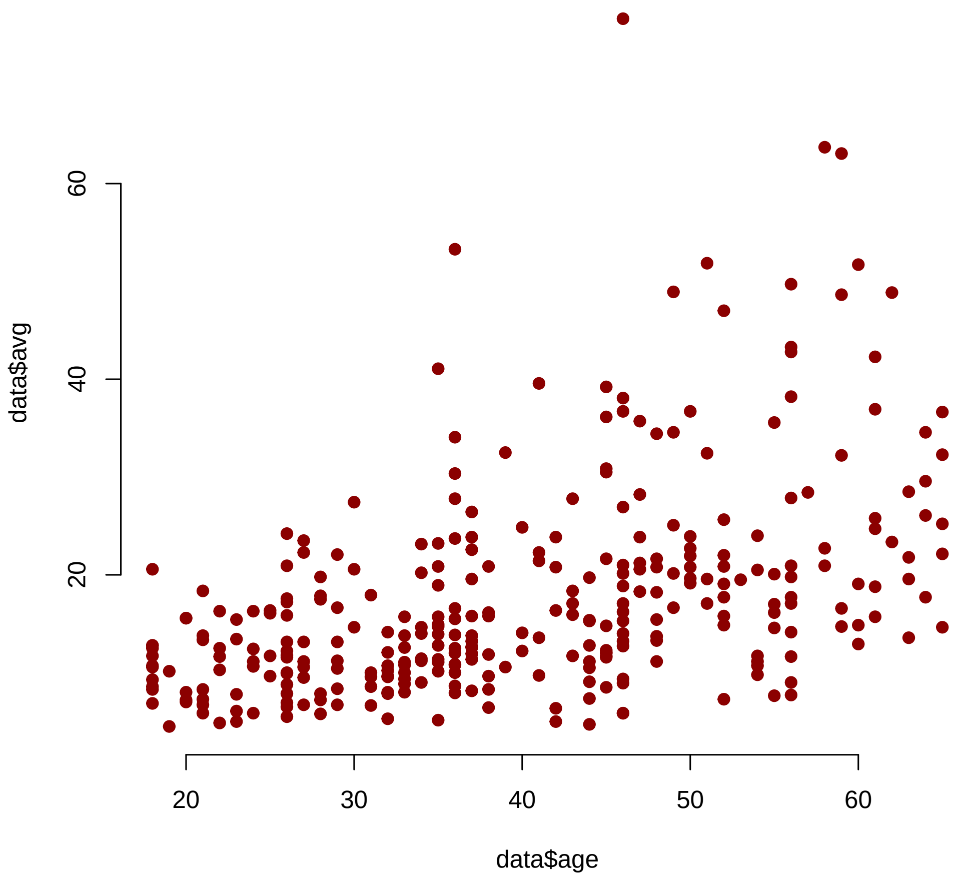
which produces several scatterplots of the three variables in the datset. To plot only two of the variables together, say `avg` and `age`, just type

```r
plot(data$age, data$avg)
```



We can improve the look of the plot by adding a few more arguments, here is an example where we change the type of points that the plot uses, and the color of the points, while we get rid of the box around the plot:

```r
plot(data$age, data$avg, type ='p', pch = 19,
    col ='dark red', frame.plot = FALSE)
```



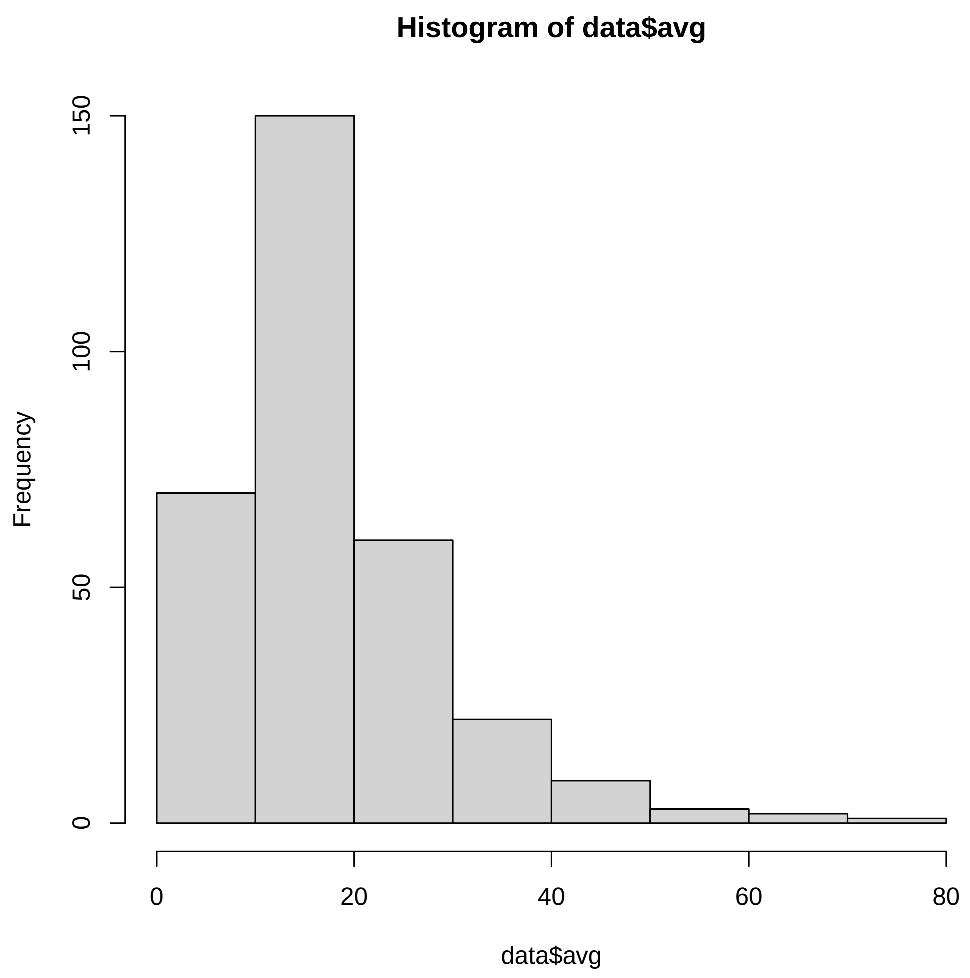
The "type" argument can take several values, here, it specifes that we want the plot to have only points (the default). With some datasets you will notice that some of the points might not be connected; this is because the data has some "NA"s, this is, some missing records either in the X or the Y variable. When there are "NA"s in the sequence, the plot function skips the connection between the points. To find out more about which arguments you can use with function plot, type the following command

```
help(plot.default)
```

This will open the help window. Now let's explore a few more plotting functions.

A very useful way of visualizing data is by using histograms, which we can do by using the "hist" function:

```
hist(data$avg)
```

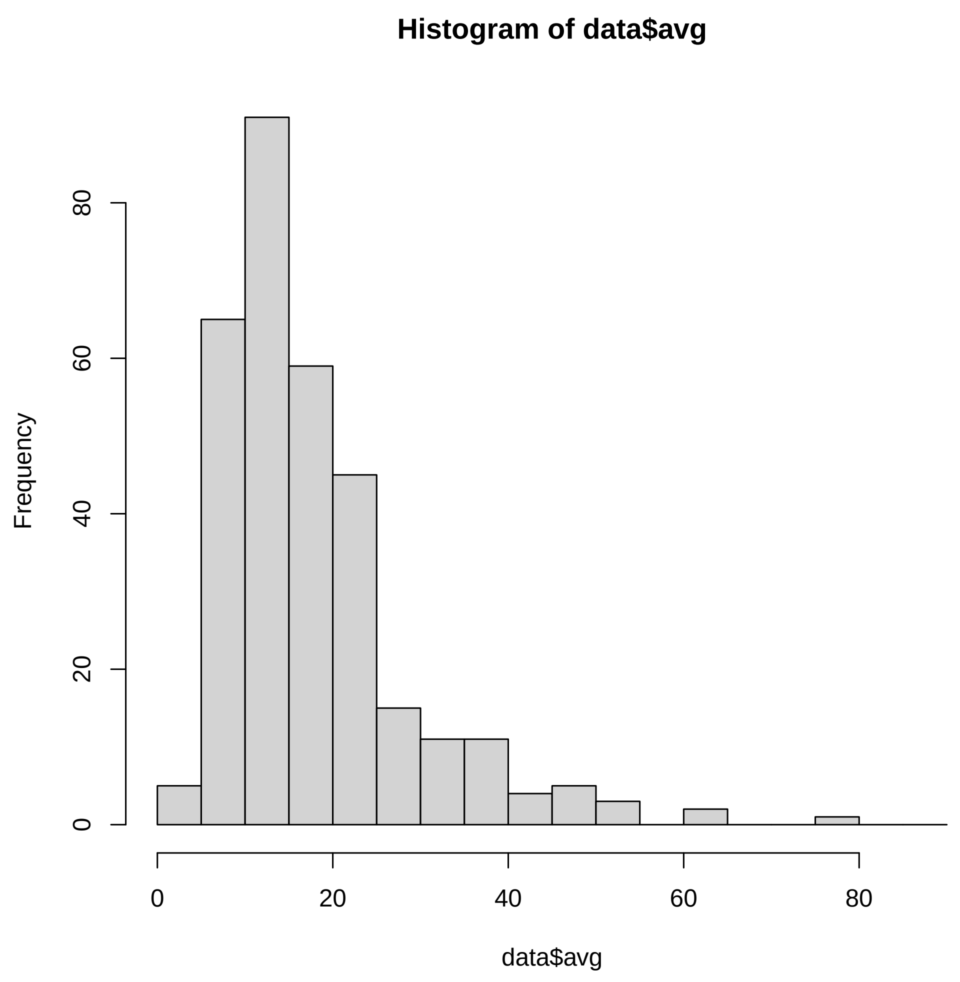


This histogram shows the frequency corresponding to different dB-intervals . Let's note the size intervals as z_i where $i = 1, 2, \ldots$. Thus, the first bar corresponds to the number of individuals between 0 and 10, this is $z_1 = [0, 10)$, the second is those individuals within the interval $z_2 = [10, 20)$ and so on. To know the range of values our size data includes, you can use the following function:

```r
range(data$avg)

## [1]  4.50000 76.85714
```

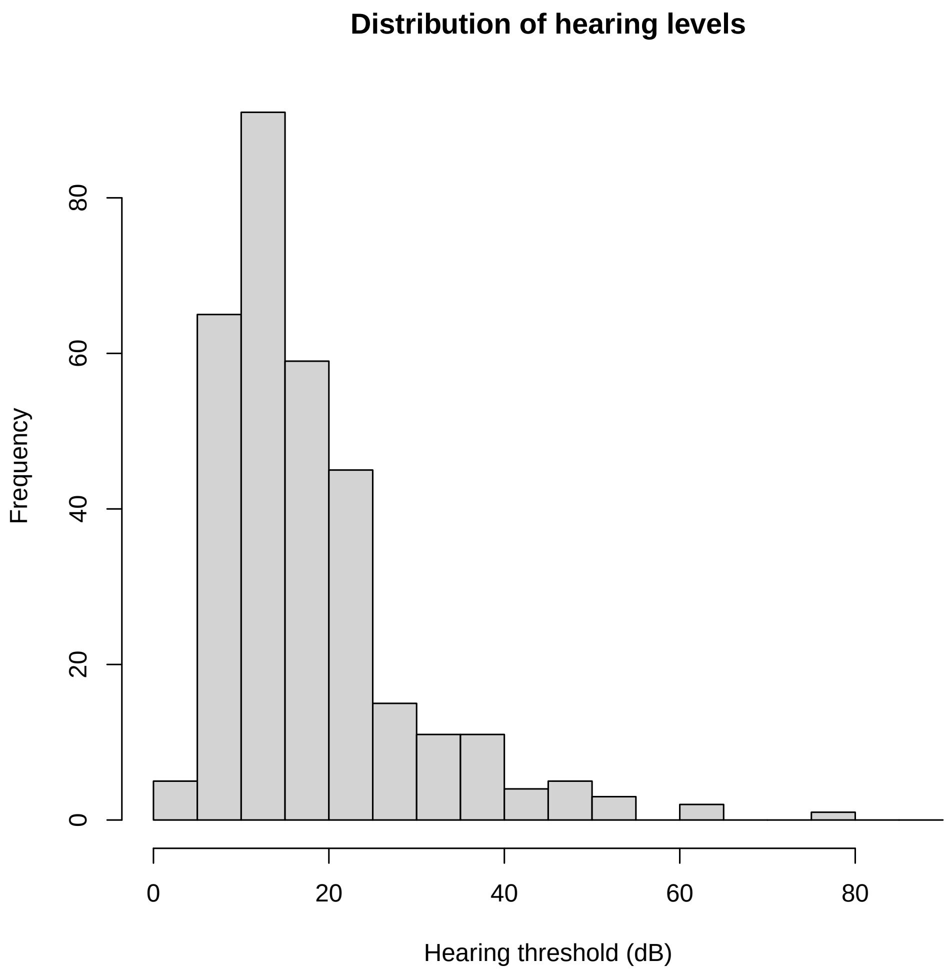
which shows the minimum and maximum values in the Fat column. Knowing this, we can change the size intervals that are plotted on the histogram by including the function "seq()" with the "breaks" argument as:

```r
hist(data$avg, breaks = seq(from = 0, to = 90, by = 5))
```



We can make this histogram look a bit better by changing the labels on the axes as well as including a more informative title. Here are the additional arguments required to do this

```r
hist(data$avg, breaks = seq(from = 0, to = 90, by = 5),
xlab = "Hearing threshold (dB)", ylab = "Frequency",
main = "Distribution of hearing levels")
```



Arguments "xlab", "ylab" and "main" can be also used with function "plot". These are in fact generic arguments applicable to most of the plotting functions in R.

The plotting functions you have been using so far belong to R's base graphics tools. An alternative popular add-on package for elaborate graphics taking care of plot details is `https://ggplot2.tidyverse.org/`.

7 Creating your own functions

One of the most attractive features on any programming language is the power to construct your own functions. Let's say that we want to construct a linear function of the form

$$\begin{aligned} f : \mathbb{R} &\rightarrow \mathbb{R} \\ f(x) &= ax + b \end{aligned}$$

where $a, b \in \mathbb{R}$. To construct this function in R we only have to choose a name for the function, say "f" and type the following code

```
f <- function(x, a, b){
a * x + b
}
```

The first part of the code line provides the name of the function object, the "<-" sign implies equality, and then we specify that the object "f" will be a function whose arguments will be "x" for the variable and "a" and "b" for the coefficients. Next, we open curly braces and, anything inside them corresponds to the operations that the function needs to do with the arguments provided. Now, as you type the code, you will notice that nothing really happens. To make the function work, you have to write it now in function form. For example, let's create an object "x" that is a sequence of values from 1 to 10 using the notation we have learned so far:

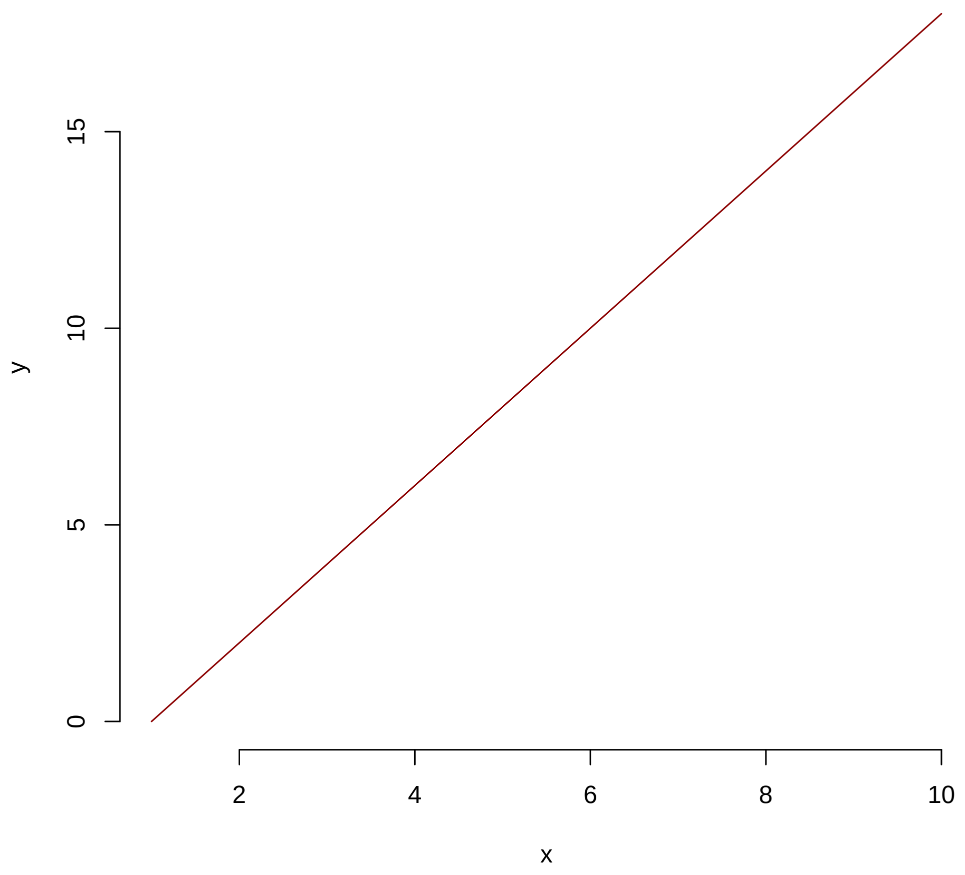
```
x=1:10
```

Now let's create a variable y by applying the function f to x and let's assume that $a = 2$ and $b = -2$

```
y=f(x=x, a=2, b=-2)
```

Now let's plot the results to see how they look like:

```
plot(x, y, type ='l', col = 'dark red', frame.plot = FALSE)
```



8 Basics of regression

For this section we will explore data from two variables, X and Y , for which we want to test if they are linearly related. These two variables can be any quantities of interest for a particular questions, such as the arrival date of seabirds to the breeding ground and the nearby temperature of the ocean, the change in a country's unemployment rate and the change in its gross domestic product, etc. Typically, there is an intuition on which of the two variables influences the values for the other one. In this case, we will take variable X as the predictor or independent variable, and Y as the dependent variable. Furthermore, we have assumed that the relationship between these two variables should be essentially linear. This means that the general trend for an observation i in the relationship between x_i with the corresponding y_i, can be represented with the linear function

$$y_i = \underbrace{\beta_0 + \beta_1 x_i}_{\hat{y}_i=\mu(x_i)} + \underbrace{\varepsilon_i}_{residuals} , \tag{1}$$

where $\beta_0, \beta_1 \in \mathbb{R}$ are the intercept and the slope coefficients of the function $\mu(x)$, ε_i is the distance between the points (x_i, y_i) and $(x_i, \hat{y}_i)$, where $\hat{y}_i$ is the value of Y predicted by the line $\mu(x)$ for observation i.

Let's take the `age` column as the explanatory variable and the `avg` data as the dependent variable. For simplicity, extract the values for each of the two variables in their own objects by typing

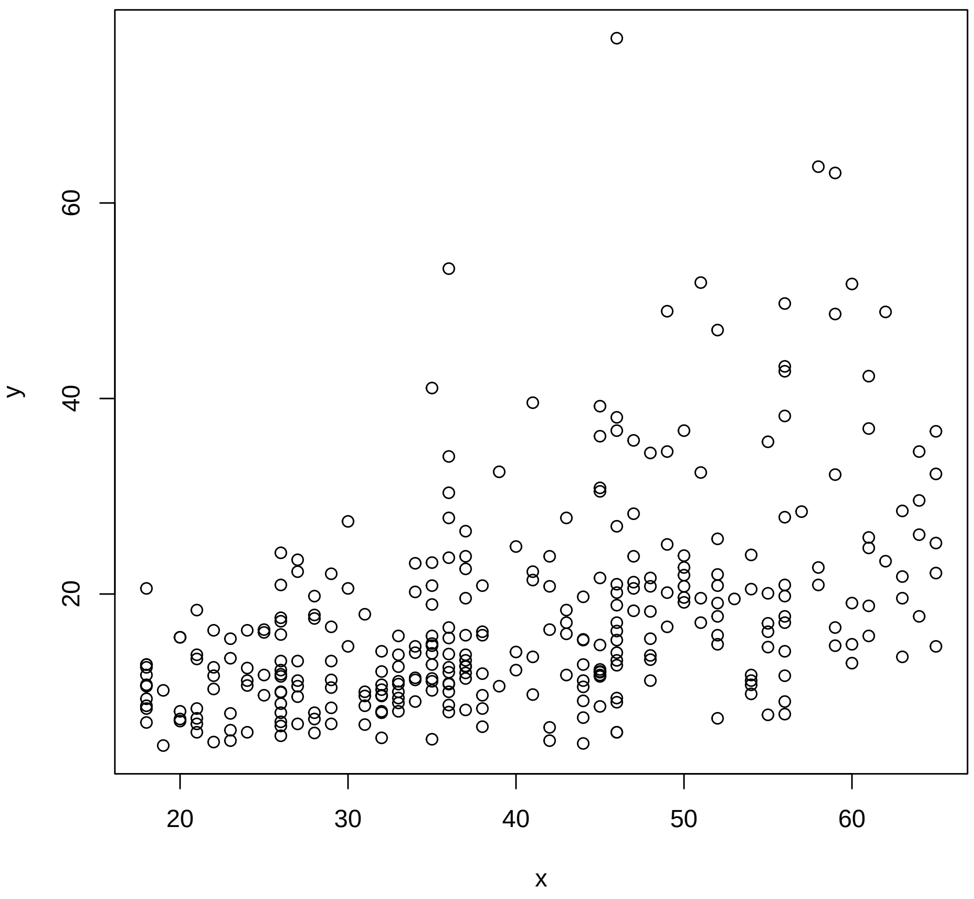
```
x <- data$age
y <- data$avg
```

and extract the sample size n by calculating the length of one of these two objects

```
n <- length(x)
```

Now let's verify that the data are really linearly related. We can do this by plotting the data, using the following code

```
plot(x,y)
```



which produces the plot below.

The scatter plot indicates an increase of hearing thresholds with age. In a first attempt to investigate this effect, we use a simple linear regression model with an intercept and a slope. In order to fit this model to our data, we need to find those values for coefficients β_0 and β_1 that would produce the line that fits most closely the data. To do this, it turns out that we need to calculate the empirical mean for both observed variables, namely $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$, with

```
xbar <- mean(x)
ybar <- mean(y)
```

Now that we have the means, let's calculate the sums of squares for X and Y , namely

$$\begin{aligned} SS_X &= \sum_{i=1}^{n}(x_i - \bar{x})^2 \\ SS_Y &= \sum_{i=1}^{n}(y_i - \bar{y})^2 \end{aligned}$$

with the following code

```
SSx <- sum((x - xbar)^2)
SSy <- sum((y - ybar)^2)
```

and the sum of squares of the crossproduct between X and Y,

$$SS_{XY} = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

with

```
SSxy <- sum((x - xbar)*(y-ybar))
```

As we will see further during this course, the slope parameter for the linear model that fits the data closest can be calculated as

$$\hat{\beta}_1 = \frac{SS_{XY}}{SS_X}$$

which can be translated into the code

```
b1hat <- SSxy / SSx
b1hat

## [1] 0.4044735
```

The corresponding estimate of the intercept parameter is then given by

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x},$$

with the code

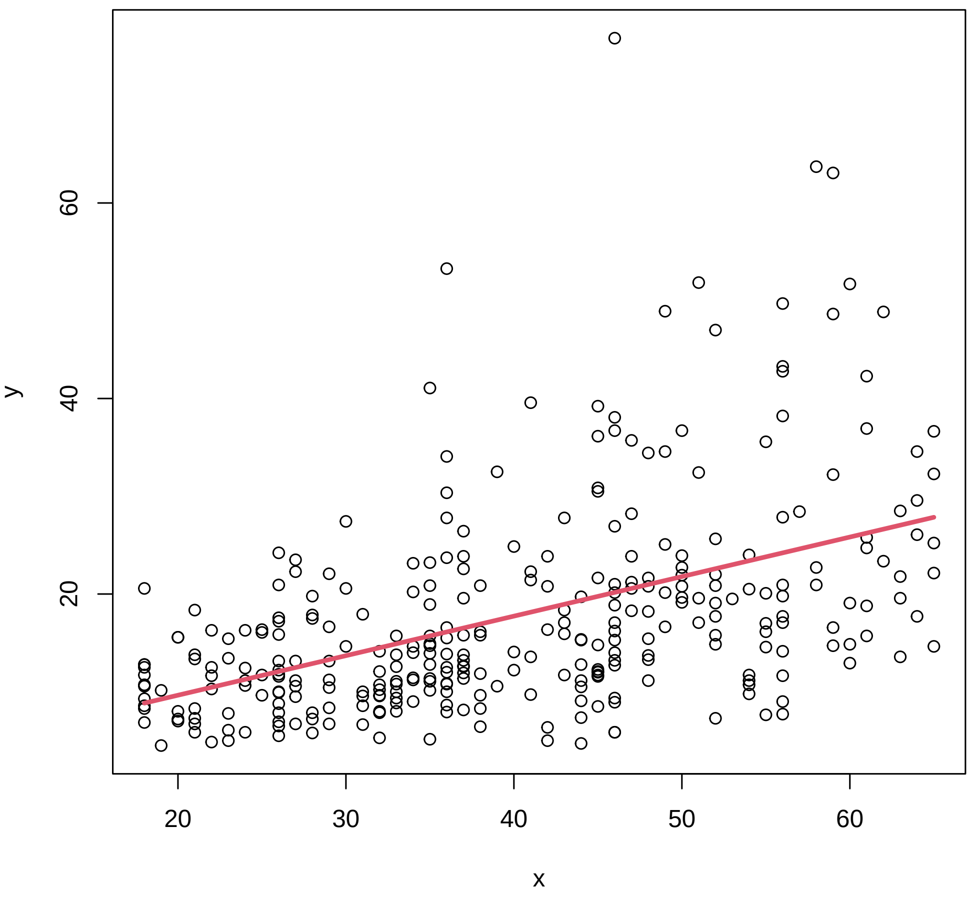
```
b0hat <- ybar - b1hat * xbar
b0hat

## [1] 1.55923
```

Now we can construct function $\mu(x)$, from which we will calculate the values for each $\hat{y}_i$, using the estimated coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$ as

```
# Function for mu(x):
mux <- function(x){
b0hat + b1hat * x
}
# Calculate the estimated y-hat:
yhat <- mux(x)
```

Let's see how well the line fits the data, with the following code

```
plot(x, y)
lines(range(x), mux(range(x)), col = 2, lwd = 3)
```



Now we can calculate the residuals, ε_i, as

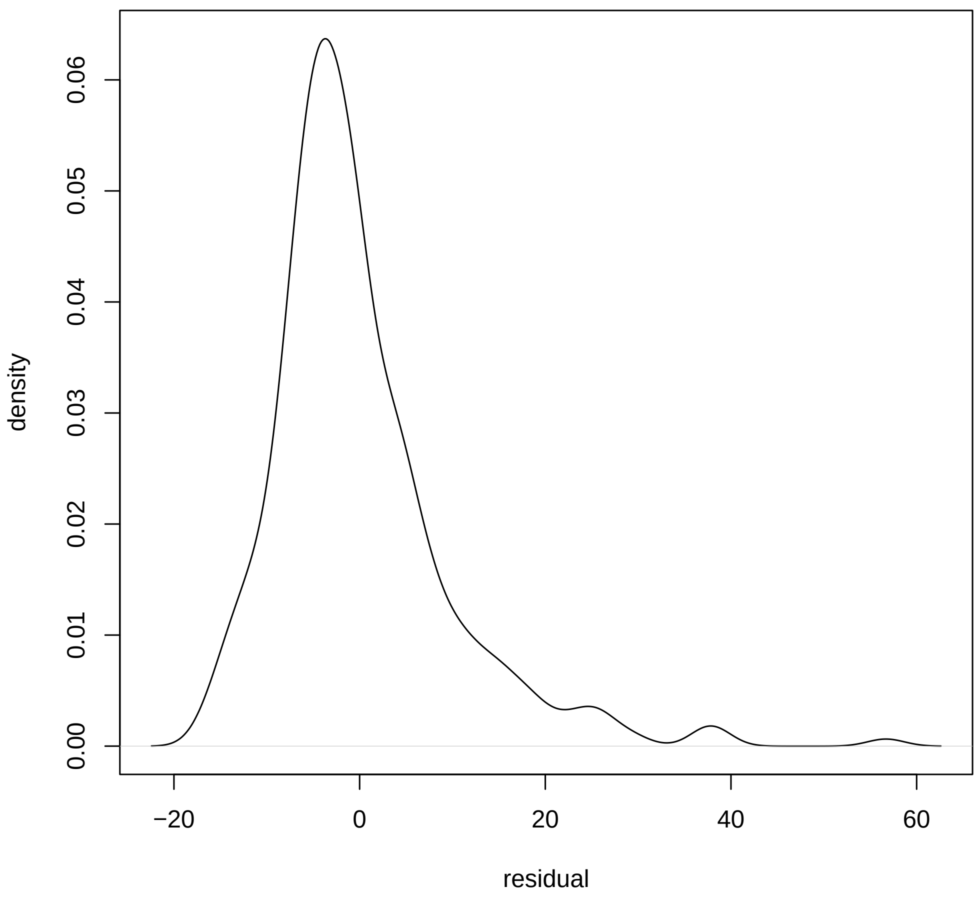
$$\varepsilon_i = y_i - \hat{y}_i,$$

with the code

```
eps <- y - yhat
```

We can visually investigate the distribution of the residuals with the function

```r
plot(density(eps), xlab = "residual", ylab = "density",
main = "")
```



Now, lastly we calculate the **residual sum of squares**, equal to

$$SS_\varepsilon = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \varepsilon_i^2,$$

which measures the amount of dispersion between the fitted line, represented by $\hat{y}_i$, and the observed values of y_i. Here is the code

```
SSe <- sum(eps^2)
SSe

## [1] 29048.96
```

With this, we can calculate the coefficient of determination R^2 as

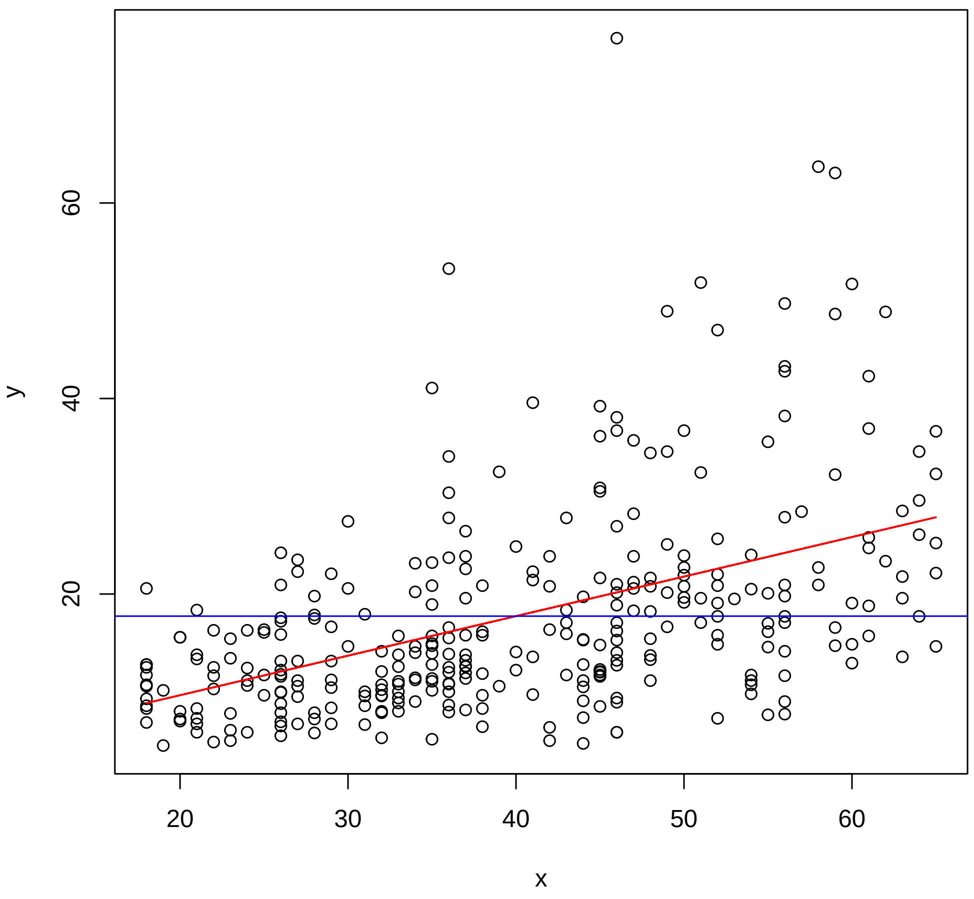
$$R^2 = 1 - \frac{SS_\varepsilon}{SS_Y},$$

where $R^2 \in [0,1]$, which provides a measure of the amount of variation in Y explained by the function $\mu(x)$, with the code

```
Rsq <- 1 - SSe / SSy
Rsq

## [1] 0.2243219
```

A value of $R^2 = 0$ implies that our fitted model is not closer to the observed data than a horizontal line passing through $\bar{y}$ (i.e. $SS_\varepsilon = SS_Y$), in other words there is no indication for an association between X and Y. A value of $R^2 > 0$ means that SS_ε is smaller than SS_Y, which implies that the data are closer to the line than to the horizontal line at $\bar{y}$ (see figure below). (Q: And which situation does a $R^2 = 1$ correspond to?)

```
plot(x,y)
lines(x,yhat, col='red')
abline(h=ybar, col='blue')
```



This is just a taste so that you can have a general idea of how you can use R to visualise and analyse data. As you can imagine, there are different ways to achieve what we did today. During the course, we will explore some other approaches and built-in functions.