Statistical Models in R

Vittorio Zampinetti[1] Mauro Gasparini[2]

2024-07-04

[1]Politecnico di Torino
[2]Politecnico di Torino

Contents

About

The book is built upon the course in Statistical Models at Politecnico di Torino, but it is self-contained and thus is accessible by any interested reader. For any feedback, send me an email: zampinetti@gmail.com. Thank you!

Chapter 1

Introduction

1.1 R and RStudio

- **R**: programming language, tailored for statisticians
- **RStudio**: development environment, software for editing and running R (and Python) applications

  It's going to change name soon, see Posit

1.1.1 Installation

First install R programming language (from here), then RStudio environment (from here).

1.2 R syntax

Here a brief overview of the main components of the R language.

Note: this is not a complete guide on the language nor on programming in general, but rather a quick-start introduction with the most used operations and structures, which assumes some very basic programming skills. For all the rest, Google is your friend (it really is).

1.2.1 Basic operations

R can be used as a fully featured calculator, typically directly from the console (see bottom panel in RStudio).

Some examples:

```
2 + 2 # calculator-style
```

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

```
c(1, 2, 3) * 4 # vector operations
```

```
## [1]  4  8 12
```

```
t(matrix(c(c(1, 2, 3, 4)),
         nrow = 2, ncol = 2)) # matrix operations (transpose)
```

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

As any other programming language, it allows for variables declaration. Variables hold a value which can be assigned, modified, and used for other computations.

```
a <- 2 # assign values to variables
b = a + 2 # equal sign also valid, but...
b
```

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

The arrow sign `<-` is the traditional sign for the assignment operation, but also `=` works. Check this examples to see why `<-` is recommended.

Conditional statements and loops have straightforward syntax (similar to C/C++ , but more compact).

```
if (b == 4 && a > 0) { # if-else statement
  out <- ""
  for (i in 1:b) { # for loop
    out <- paste(out, class(b)) # concatenate two or more strings
  }
  print(out) # print string to output
} else {
  stop("error") # stop program and exit
}
```

```
## [1] " numeric numeric numeric numeric"
```

While `<-` is only used for variable assignment, i.e. filling up the variable allocated space with some value, the `=` is both used for assignment and function argument passing (as introduced in Functions).

1.3 Functions

Functions are called with the `()` operator and present named arguments.

```r
# get Normal samples with 0 (default) mean and 10 standard deviation
array <- rnorm(10, sd = 10) # sd is an argument,
                            # the mean is left to the default 0
array
```

```
##  [1]   3.481589 -14.993989  -0.263854 -10.533974  -9.974914  -4.220047
##  [7]   9.341907   8.410405  -5.788265   5.764697
```

You can explicitly call the arguments to which you want to pass the parameters (e.g. sd of the Gaussian). In this case it is necessary, because the second position argument is the mean, but we want it to be the default.

1.3.1 Definition

To define a custom function

```r
# implements power operation
#'
#' @param x a real number
#' @param n a natural number, defaults to 2
#' @return n-th power of x
pow <- function(x, n = 2) {
  ans <- 1
  if (n > 0) {
    for (i in 1:n) {
      ans <- ans * x
    }
  } else if (n == 0) {
    # do nothing, ans is already set to 1
  } else {
    stop("error: n must be non-negative")
  }
  return(ans)
}

print(paste("3^5 = ", pow(3, 5),
            ", 4^2 = ", pow(4),
            ", 5^0 = ", pow(5, 0)))
```

```
## [1] "3^5 =  243 , 4^2 =  16 , 5^0 =  1"
```

The `return()` call can be omitted (but it's better not to).

```r
pow <- function(a, b) {
  ans <- 1
  # ...
  # last expression is returned
```

```
  ans # this is equivalent to write return(ans)
}
```

1.4 Data types

Every variable in R is an object, which holds a value (or collection of values) and some other attributes, properties or methods.

1.4.1 Base types

There are 5 basic data types:

- **numeric** (real numbers)
- **integer**
- **logical** (aka boolean)
- **character**
- **complex** (complex numbers)

1.4.1.1 Numeric

When you write numbers in R, they are going to default to numeric type values

```
a <- 3.4 # decimal
b <- -.30 # signed decimal
c <- 42 # also without dot, it's numeric

print(paste0("data types | a:", class(a),
            ", b:", class(b),
            ", c:", class(c)))
```

```
## [1] "data types | a:numeric, b:numeric, c:numeric"
```

1.4.1.2 Integer

Integer numbers can be enforced typing an `L` next to the digits. Casting is implicit when the result of an operation involving integers is not an integer

```
int_num <- 50L # 50 stored as integer

non_casted_num <- int_num - 2L # result is exactly 48, still int
casted_num <- int_num / 7L # implicit cast to numeric type to store decimals
print(paste(class(int_num), class(non_casted_num),
            class(casted_num), sep = ", "))
```

```
## [1] "integer, integer, numeric"
```

1.4.1.3 Logical

The `logical` type can only hold `TRUE` or `FALSE` (in capital letters)

```
bool_a <- FALSE
bool_b <- T # T and F are short for TRUE and FALSE
bool_a | !bool_b # logical or between F and not-T
```

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

You can test the value of a boolean also using `0,1`

```
bool_a == 0 # if 'A' is not equal to 'not B', raise an error
```

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

and a sum between logical values, treats `FALSE = 0` and `TRUE = 1`, which is useful when counting true values in logical arrays

```
bool_a + bool_b + bool_b # FALSE + TRUE + TRUE (it's not an OR operation)
```

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

1.4.1.4 Character

A character is any number of characters enclosed in quotes `' '` or double quotes `" "`.

```
char_a <- "," # single character
char_b <- "bird" # string
char_c <- 'word' # single quotes
full_char <- paste(char_b, char_a, char_c) # concatenate chars
class(full_char) # still a character
```

```
## [1] "character"
```

1.4.1.5 Complex

```
complex_num <- 5 + 4i
Mod(complex_num) # try all complex operations, e.g. modulus
```

```
## [1] 6.403124
```

1.4.1.6 Special values

- `NA`: "not available", missing value
- `Inf`: infinity
- `NaN`: "not-a-number", undefined value

```
missing_val <- NA
is.na(missing_val) # test if value is missing
```

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

Every operation involving missing values, will output `NA`

```
missing_val == NA # cannot use ==
```

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

```
print(paste(
  "1/0 = ", 1 / 0,
  ", 0/0 = ", 0 / 0
))
```

```
## [1] "1/0 =  Inf , 0/0 =  NaN"
```

These special values are not necessarily unwanted, but they require extra care. E.g. `Inf` can appear also in case of numerical *overflow*.

```
exp(1000)
```

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

1.4.1.7 Conversion

Variables types can be converted with `as.<typename>()`-like functions, as long as conversion makes sense. Some examples:

```
v <- TRUE
w <- "0"
x <- 3.2
y <- 2L
z <- "F"
cat(paste(
  paste(x, as.integer(x), sep = " => "), # from numeric to integer
  paste(y, as.numeric(y), sep = " => "), # from integer to numeric
  paste(y, as.character(y), sep = " => "),  # from integer to character
  paste(w, as.numeric(w), sep = " => "),  # from number-char to numeric
  paste(v, as.numeric(v), sep = " => "), # from logical to numeric
  sep = "\n"
))
```

```
## 3.2 => 3
## 2 => 2
## 2 => 2
## 0 => 0
## TRUE => 1
```

```
as.numeric(z)  # from character to numeric (coercion warning - NA)
```

```
## Warning: NAs introduced by coercion
```

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

1.4.2 Vectors and matrices

1.4.2.1 Vectors

Vectors are build with the `c()` function. A vector holds values of the same type.

```
vec1 <- c(4, 3, 9, 5, 8)
vec1
```

```
## [1] 4 3 9 5 8
```

Vector operations and aggregation of values is as intuitive as it can be.

```
vec2 <- vec1 - 1 # subtract 1 to all values (broadcast)
sum(vec1) # sum all values in vec1
```

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

```
mean(vec2) # compute the mean
```

```
## [1] 4.8
```

```
sort(vec1, decreasing = TRUE) # sort elements in decreasing order
```

```
## [1] 9 8 5 4 3
```

Conversion is still possible and it's made element by element.

```
char_vec <- as.character(vec1) # convert every value in vec1 to char
char_vec
```

```
## [1] "4" "3" "9" "5" "8"
```

Range vectors (unit-stepped intervals) are built with `start:end` syntax. Note: the type of range vectors is `integer`, not `numeric`.

```
x_range <- 1:10
class(x_range)
```

```
## [1] "integer"
```

They are particularly useful in loops statements:

```
vec3 <- c() # declare an empty vector
# iterate all the indices along vec1
for (i in 1:length(vec1)) {
  vec3[i] <- vec1[i] * i # access with [idx]
}
vec3
```

```
## [1]  4  6 27 20 40
```

Vector elements are selected with square brackets `[]`. Putting vectors inside brackets performs slicing

```
vec1[1:3] # first 3 elements
```

```
## [1] 4 3 9
```

```
vec1[c(1, 3)] # only first and third element
```

```
## [1] 4 9
```

```
vec1[-c(1:3)] # all but elements 1 to 3
```

```
## [1] 5 8
```

```
vec1[seq(1, length(vec1), 2)] # odd position elements
```

```
## [1] 4 9 8
```

To find an element in a vector and get its index/indices, the `which()` function can be used

```
which(vec1 == 3)
```

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

```
which(vec1 < 5)
```

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

And finally, to filter only values that satisfy a certain condition, we can combine `which` with splicing.

```
vec1[which(vec1 >= 5)]
```

```
## [1] 9 5 8
```

```
# or, equivalently, using logical masking
vec1[vec1 >= 5]
```

```
## [1] 9 5 8
```

1.4.2.2 Matrices

Matrices are built with `matrix()`

```
mat1 <- matrix(1:24,
               nrow = 6, ncol = 4)
mat1 # filled column-wise (default)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    7   13   19
## [2,]    2    8   14   20
## [3,]    3    9   15   21
```

```
## [4,]    4   10   16   22
## [5,]    5   11   17   23
## [6,]    6   12   18   24
```

```
mat2 <- matrix(1:24,
               nrow = 6, ncol = 4, byrow = TRUE)
mat2 # filled row-wise
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
## [5,]   17   18   19   20
## [6,]   21   22   23   24
```

```
dim(mat2) # get dimensions
```

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

```
c(nrow(mat2), ncol(mat2)) # get number of rows and cols separately
```

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

```
# or, equivalently
dim(mat2)[1] # nrow
```

```
## [1] 6
```

All indexing operations available on vectors, are also available on matrices

```
mat2[1, 1] # element 1,1
```

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

```
mat2[3, ] # third row (empty space for all elements)
```

```
## [1]  9 10 11 12
```

```
mat2[1:2, 1:2] # upper left 2x2 sub-matrix
```

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

```
t(mat2) # transposed matrix
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    5    9   13   17   21
## [2,]    2    6   10   14   18   22
## [3,]    3    7   11   15   19   23
## [4,]    4    8   12   16   20   24
```

Operations with matrix and vectors can be both element-wise and matrix operations (e.g. scalar product). Note that a vector built with `c()` is a column vector by default. Some examples:

```
diagonal_mat <- diag(nrow = 4) # 4x4 identity matrix
 # element by element
diagonal_mat * 1:2 # note: 1:2 is repeated to match the matrix dimensions
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    2    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    2
```

```
diagonal_mat %*% seq(2, 8, 2) # matrix multiplication (4,4) x (4, 1) -> (4, 1)
```

```
##      [,1]
## [1,]    2
## [2,]    4
## [3,]    6
## [4,]    8
```

```
v1 <- 1:4
v2 <- 4:1
v1 %*% v2 # here v1 is implicitly converted to row vector
```

```
##      [,1]
## [1,]   20
```

1.4.2.3 Arrays

Arrays are multi-dimensional vectors (generalization of a matrix with more than two dimensions). They work pretty much like matrices.

```
arr1 <- array(1:24, dim = c(2, 4, 3))
arr1
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]    9   11   13   15
## [2,]   10   12   14   16
##
```

```
## , , 3
##
##      [,1] [,2] [,3] [,4]
## [1,]   17   19   21   23
## [2,]   18   20   22   24
```

```r
arr1[2, 1, 3] # get one element
```

```
## [1] 18
```

```r
sliced_arr <- arr1[, 2, ] # slice at column 2
sliced_arr
```

```
##      [,1] [,2] [,3]
## [1,]    3   11   19
## [2,]    4   12   20
```

```r
dim(sliced_arr) # reduces ndims by one (dimension selected is dropped)
```

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

1.4.3 Lists and dataframes

Lists are containers that can hold different data types. Each entry, which can even be another list, has a position in the list and can also be named.

```r
list1  <- list(1:3, TRUE, x = c("a", "b", "c"))
list1
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] TRUE
##
## $x
## [1] "a" "b" "c"
```

```r
list1[[3]] # access with through index
```

```
## [1] "a" "b" "c"
```

```r
list1$x # access through name
```

```
## [1] "a" "b" "c"
```

Dataframes are collections of columns that have the same length. Contrarily to matrices, columns in dataframes can be of different types. They are the most common way of representing structured data and most of the dataset will be stored in dataframes.

```
df1 <- data.frame(x = 1, y = 1:10,
           char = sample(c("a", "b"), 10, replace = TRUE))

df1 # x was set to just one value and gets repeated ('recycled')
```

```
##    x  y char
## 1  1  1    a
## 2  1  2    a
## 3  1  3    a
## 4  1  4    b
## 5  1  5    a
## 6  1  6    a
## 7  1  7    b
## 8  1  8    b
## 9  1  9    b
## 10 1 10    a
```

```
df1[[2]] # access through column index
```

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

```
df1$x # access through column name
```

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

```
df1[, 3] # access with matrix-style index
```

```
##  [1] "a" "a" "a" "b" "a" "a" "b" "b" "b" "a"
```

```
df1[2:4, ] # can also select subset of rows
```

```
##   x y char
## 2 1 2    a
## 3 1 3    a
## 4 1 4    b
```

The `dplyr` library provides another dataframe object (called *tibble*) which has all the effective features of Base R `data.frame` and none of the deprecated functionalities. It's simply a newer version of dataframes (therefore recommended over the old one).

```
library("tibble")
tibble(x = 1:15, y = 1, z = x / y) # tibble dataframe
```

```
## # A tibble: 15 x 3
##        x     y     z
##    <int> <dbl> <dbl>
##  1     1     1     1
##  2     2     1     2
```

```
##   3     3     1     3
##   4     4     1     4
##   5     5     1     5
##   6     6     1     6
##   7     7     1     7
##   8     8     1     8
##   9     9     1     9
##  10    10     1    10
##  11    11     1    11
##  12    12     1    12
##  13    13     1    13
##  14    14     1    14
##  15    15     1    15
```

For more information on tibble and its advantages with respect to traditional dataframes, type `vignette("tibble")` in an R console. Notice that you can convert datasets to tibble with `as_tibble()`, while with `as.data.frame()` you will get a Base R dataframe.

1.5 Data manipulation

Now that we know what a dataframe is and how it is generated, we can focus on data manipulation.

The `dplyr` library provides an intuitive way of working with datasets. For instance, let's consider the `mtcars` dataset.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
mtcars$modelname <- rownames(mtcars) # name column with models
mtcars <- as_tibble(mtcars) # convert to tibble
mtcars # display the raw data
```

```
## # A tibble: 32 x 12
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb modelname
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1  21       6  160    110  3.9   2.62  16.5     0     1     4     4 Mazda RX4
```

```
##  2  21        6  160     110  3.9    2.88  17.0     0     1     4     4 Mazda RX4 ~
##  3  22.8      4  108      93  3.85   2.32  18.6     1     1     4     1 Datsun 710
##  4  21.4      6  258     110  3.08   3.22  19.4     1     0     3     1 Hornet 4 D~
##  5  18.7      8  360     175  3.15   3.44  17.0     0     0     3     2 Hornet Spo~
##  6  18.1      6  225     105  2.76   3.46  20.2     1     0     3     1 Valiant
##  7  14.3      8  360     245  3.21   3.57  15.8     0     0     3     4 Duster 360
##  8  24.4      4  147.     62  3.69   3.19  20       1     0     4     2 Merc 240D
##  9  22.8      4  141.     95  3.92   3.15  22.9     1     0     4     2 Merc 230
## 10  19.2      6  168.    123  3.92   3.44  18.3     1     0     4     4 Merc 280
## # i 22 more rows
```

Let's say we want to get the cars with more than 100 hp, and we are just interested in the car model name and we want the data to be sorted in alphabetic order.

```r
mtcars %>% # send the data into the transformation pipe
  dplyr::filter(hp > 100) %>% # filter rows with hp > 100
  dplyr::select(modelname) %>% # filter columns (select only modelname col)
  dplyr::arrange(modelname) # display in alphabetic order
```

```
## # A tibble: 23 x 1
##    modelname
##    <chr>
##  1 AMC Javelin
##  2 Cadillac Fleetwood
##  3 Camaro Z28
##  4 Chrysler Imperial
##  5 Dodge Challenger
##  6 Duster 360
##  7 Ferrari Dino
##  8 Ford Pantera L
##  9 Hornet 4 Drive
## 10 Hornet Sportabout
## # i 13 more rows
```

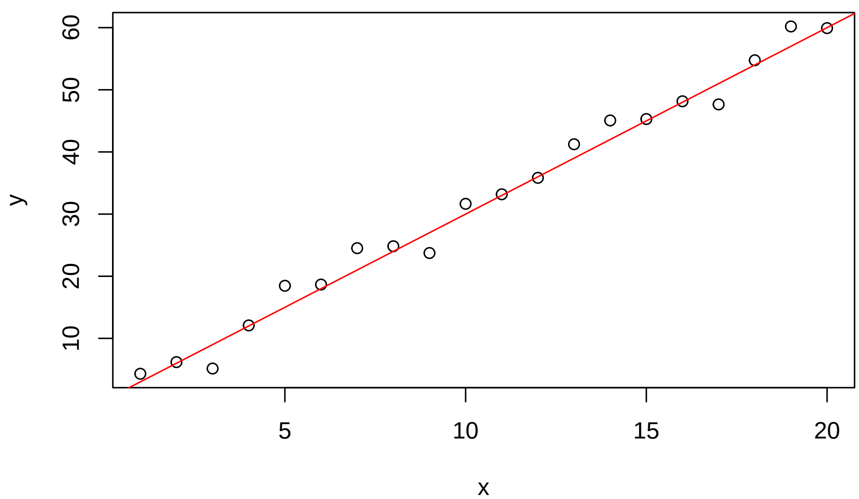
There are many other dplyr functions for data transformation. This useful cheatsheet summarizes most of them for quick access.

1.6 Plotting

In base R plots are built with several calls to functions, each of which edit the current canvas. For instance, to plot some points and a line:

```r
# generate synthetic data
n_points <- 20
x <- 1:n_points
y <- 3 * x + 2 * rnorm(n_points)
```

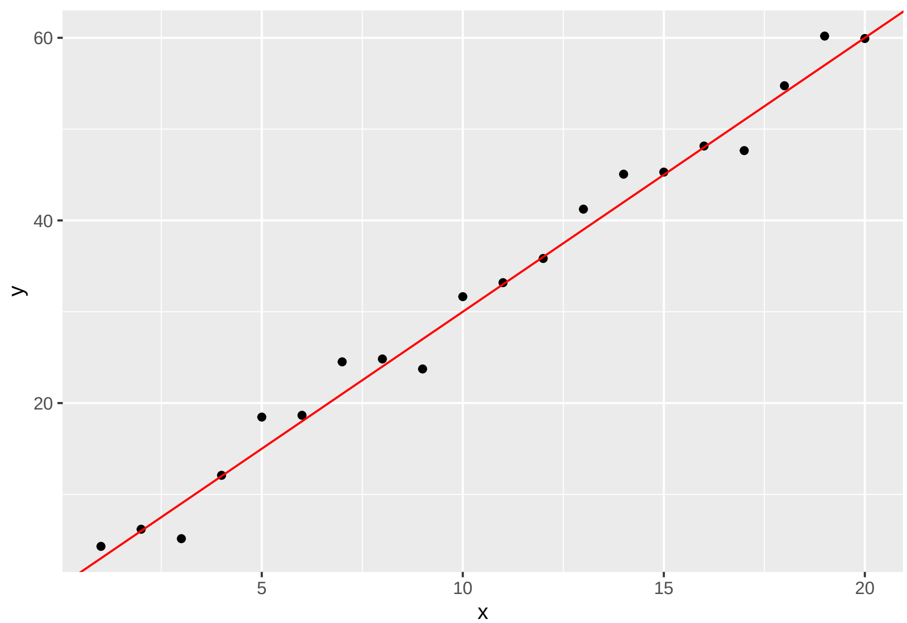
```r
plot(x, y)
abline(a = 0, b = 3, col = "red")
```



However, the `ggplot` library is now the new standard plotting library. In ggplot, a plot is decomposed in three main components: data, coordinate system and visual marks, called *geoms*. The plot is built by stacking up layers of visualization objects. Data is in form of dataframes and the columns are selected in the aesthetics arguments.

The same plot shown before can be drawn with ggplot in the following way.

```r
library(ggplot2)
gg_df <- tibble(x = x, y = y)

ggplot(gg_df) +
  geom_point(mapping = aes(x, y)) +
  geom_abline(mapping = aes(intercept = 0, slope = 3), color = "red")
```



This is just a brief example. More will be seen in the next lessons. Check out this cheatsheet for quick look-up on ggplot functions.

1.7 Examples: plot and data manipulation

Combining altogether, here a data visualization workflow on the Gapminder dataset.

```r
library(gapminder)
# have a quick look at the Gapminder dataset
str(gapminder)
```

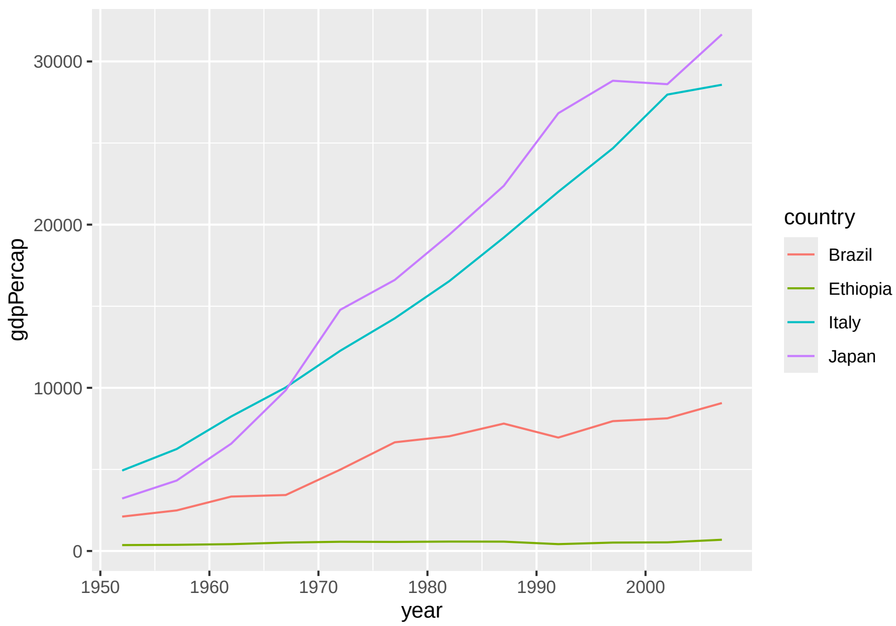
```
## tibble [1,704 x 6] (S3: tbl_df/tbl/data.frame)
##  $ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ year     : int [1:1704] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
##  $ lifeExp  : num [1:1704] 28.8 30.3 32 34 36.1 ...
##  $ pop      : int [1:1704] 8425333 9240934 10267083 11537966 13079460 14880372 12881816 1
##  $ gdpPercap: num [1:1704] 779 821 853 836 740 ...
```

A factor, which we haven't seen yet, is just a data-type characterizing a discrete categorical variable; the levels of a factor describe how many distinct categories it can take value from (e.g. the variable `continent` takes values from the set `{Africa, Americas, Asia, Europe, Oceania}`).

Let's say we want to compare the GDP per capita of some different countries

(Italy, Japan, Brasil and Ethiopia), plotted against time (year by year).

```
# transform the dataset according to what is necessary
wanted_countries <- c("Italy", "Japan", "Brazil", "Ethiopia")
gapminder %>%
  dplyr::filter(country %in% wanted_countries) %>%
  # now feed the filtered data to ggplot (using the pipe op)
  ggplot() +
    geom_line(aes(year, gdpPercap, color = country))
```



If we want to add some information about the same measure over the whole continent, showing for instance the boundaries of GDP among all countries in the same continent of the four selected countries, this is more or less what we can do

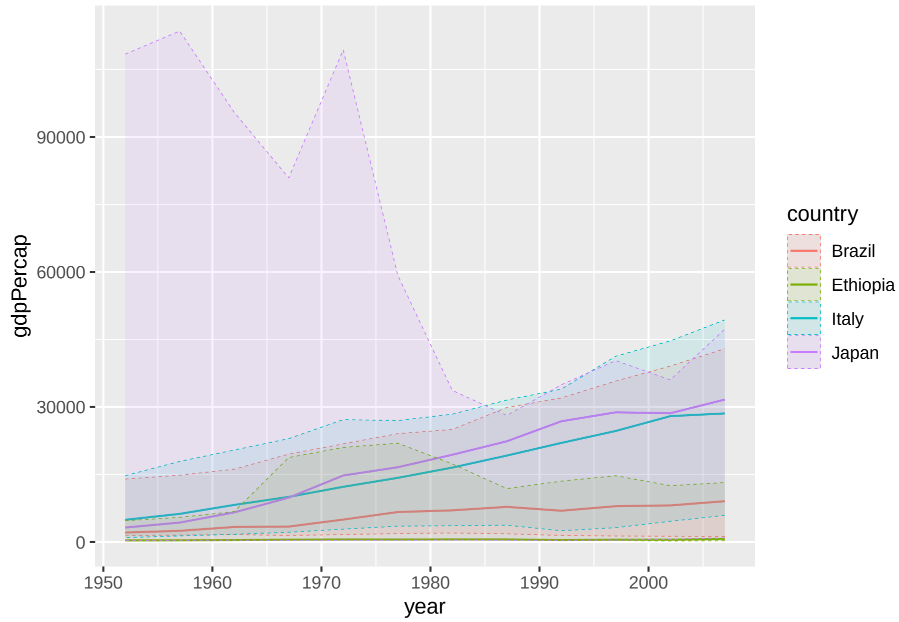
```
# give all the data to ggplot, we'll filter later
plt <- gapminder %>%
  ggplot() +
    geom_line(data = . %>%
      dplyr::filter(country %in% wanted_countries),
              aes(year, gdpPercap, color = country)) +
    # now group by continent and get the upper/lower bounds
    geom_ribbon(data = . %>%
      # min(NA) = NA, make sure NAs are excluded
      dplyr::filter(!is.na(gdpPercap)) %>%
      # gather all entries for each continent separately
```

```r
      dplyr::group_by(continent, year) %>%
      # compute aggregated quantity (min/max)
      dplyr::summarize(minGdp = min(gdpPercap),
                       maxGdp = max(gdpPercap), across()) %>%
      dplyr::filter(country %in% wanted_countries),
              aes(ymin = minGdp, ymax = maxGdp,
                  x = year, color = country, fill = country),
              alpha = 0.1, linetype = "dashed", size = 0.2)
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```r
plt
```

```
## Warning: There was 1 warning in `dplyr::summarize()`.
## i In argument: `across()`.
## Caused by warning:
## ! Using `across()` without supplying `.cols` was deprecated in dplyr 1.1.0.
## i Please supply `.cols` instead.

## Warning: Returning more (or less) than 1 row per `summarise()` group was deprecated in
## dplyr 1.1.0.
## i Please use `reframe()` instead.
## i When switching from `summarise()` to `reframe()`, remember that `reframe()`
##   always returns an ungrouped data frame and adjust accordingly.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

## `summarise()` has grouped output by 'continent', 'year'. You can override using
## the `.groups` argument.
```



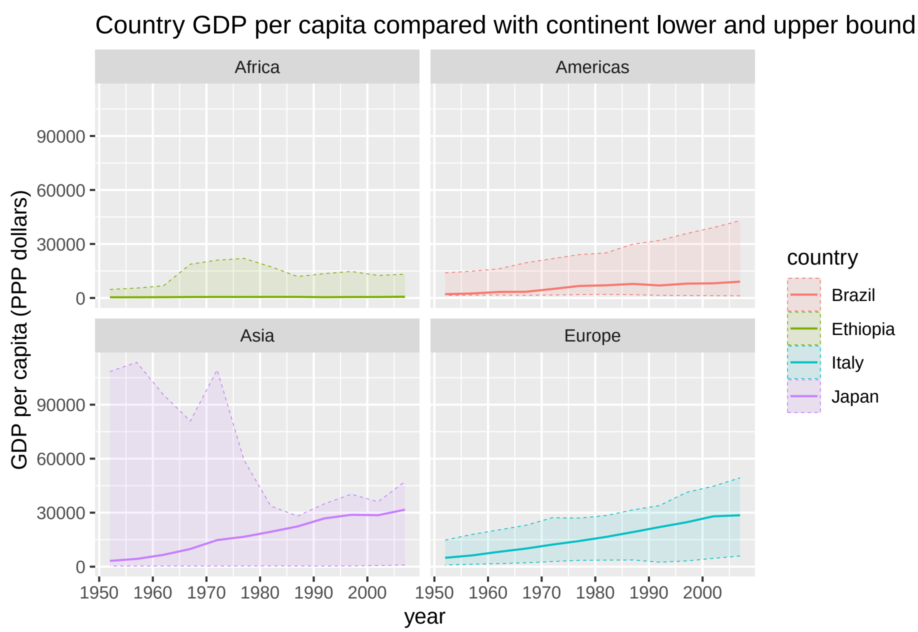
But, since it looks a bit confusing, we might want four separate plots.

```
wanted_continents <- gapminder %>%
  dplyr::filter(country %in% wanted_countries) %>%
  # extract one column from a dataframe (different from select)
  dplyr::pull(continent) %>%
  unique()

gapminder %>%
  dplyr::filter(continent %in% wanted_continents) %>%
  ggplot() +
    geom_line(data = . %>%
      dplyr::filter(country %in% wanted_countries),
              aes(year, gdpPercap, color = country)) +
    # now group by continent and get the upper/lower bounds
    geom_ribbon(data = . %>%
      # min(NA) = NA, make sure NAs are excluded
      dplyr::filter(!is.na(gdpPercap)) %>%
      # gather all entries for each continent separately
      dplyr::group_by(continent, year) %>%
      # compute aggregated quantity (min/max)
      dplyr::summarize(minGdp = min(gdpPercap),
                       maxGdp = max(gdpPercap), across()) %>%
      dplyr::filter(country %in% wanted_countries),
              aes(ymin = minGdp, ymax = maxGdp,
                  x = year, color = country, fill = country),
```

```
            alpha = 0.1, linetype = "dashed", size = 0.2) +
    facet_wrap(vars(continent)) +
    labs(title = paste("Country GDP per capita compared with",
         "continent lower and upper bounds"),
         x = "year", y = "GDP per capita (PPP dollars)")
```

```
## Warning: Returning more (or less) than 1 row per `summarise()` group was deprecated in
## dplyr 1.1.0.
## i Please use `reframe()` instead.
## i When switching from `summarise()` to `reframe()`, remember that `reframe()`
##   always returns an ungrouped data frame and adjust accordingly.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## `summarise()` has grouped output by 'continent', 'year'. You can override using
## the `.groups` argument.
```



1.8 Probability

Base R provide functions to handle almost any probability distribution. These functions are usually divided into four categories:

- density function
- distribution function
- quantile function

- random function (sampling)

```
n <- 10
normal_samples <- rnorm(n = n, mean = 0, sd = 1) # sample 10 Gaussian samples
normal_samples
```

```
##  [1]  1.1786512  0.5189134 -1.0445893 -2.8316918  0.1804755 -0.4073548
##  [7]  1.0915080 -2.1503819 -0.6723944  0.6825851
```

```
# compute the density function (over another Normal)
dnorm(normal_samples, mean = 2, sd = 1)
```

```
##  [1] 2.847210e-01 1.332208e-01 3.873098e-03 3.400591e-06 7.620919e-02
##  [6] 2.200211e-02 2.640498e-01 7.251092e-05 1.122306e-02 1.675071e-01
```

```
# cumulative distribution function
pnorm(normal_samples, mean = 0, sd = 1)
```

```
##  [1] 0.880731460 0.698089442 0.148106411 0.002315123 0.571610364 0.341873709
##  [7] 0.862475294 0.015762510 0.250666311 0.752565484
```

```
# get the quantiles of a normal
qnorm(c(0.05, 0.95), mean = 0, sd = 1)
```

```
## [1] -1.644854  1.644854
```

1.9 Extras

1.9.1 File system and helper

R language provides several tools for management of files and function help. Here some useful console commands. Note that most of them are also available on RStudio through the graphic interface (buttons).

R saves all the variables and you can display them with `ls()`.

```
rm(list = ls()) # clear up the space removing all variables stored so far
# let's add some variables
x <- 1:10
y <- x[x %% 2 == 0]

ls() # check variables in the environment
```

```
## [1] "x" "y"
```

The working directory is the folder located on your computer from which R navigates the filesystem.

```
getwd() # check your wd
```

```
## [1] "/Users/runner/work/statistical-models-r/statistical-models-r"
```

```r
setwd("./tmp") # set the working directory to an arbitrary (existing) folder
# save the current environment
save.image("./01_test.RData")
# check that it's on the working directory
dir()
```

RStudio typically save the environment automatically, but sometimes (if not every time you close R) you should clear the environment variables, because loading many variables when opening RStudio might fill up too much memory.

You can also read function helpers simply by typing `?function_name`. This will open a formatted page with information about a specific R function or object.

```r
?quit # help for the quit function

?Arithmetic # help for more general syntax information
help(Trig) # or use help(name)
```

1.9.2 Packages

Packages can be installed via command line using `install.packages("package_name")`, or through RStudio graphical interface.

```r
# the following function call is commented because package installation should
# not be included in a script (but you can find it commented, showing that the
# script requires a package as dependency)

# install.packages("tidyverse")
```

And then you can load the package with `library`.

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v readr     2.1.5
## v lubridate 1.9.3     v stringr   1.5.1
## v purrr     1.0.2     v tidyr     1.3.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to b
```

1.9.3 Arrow sign

The difference between `<-` and `=` is not just programming *style* preference. Here an example where using `=` rather than `<-` makes a difference:

```
# gives error: argument 'b' is not part of 'within'
within(data.frame(a = rnorm(2)), b = a^2)
```

```
## Error in eval(substitute(expr), e): argument is missing, with no default
```

```
# 'b<-a^2' is the value passed to the expr argument of within()
within(data.frame(a = rnorm(2)), b <- a^2)
```

```
##            a         b
## 1 -0.4801796 0.2305724
## 2  0.9055271 0.8199793
```

Although this event might never occur in one's programming experience, it's safer (and more elegant) to use `<-` when assigning variable.

Besides, `->` is also valid and it is used (more intuitive) when assigning pipes result to variables.

```
library(dplyr)

x <- starwars %>%
  dplyr::mutate(bmi = mass / ((height / 100) ^ 2)) %>%
  dplyr::filter(!is.na(bmi)) %>%
  dplyr::group_by(species) %>%
  dplyr::summarise(bmi = mean(bmi)) %>%
  dplyr::arrange(desc(bmi))

x
```

```
## # A tibble: 32 x 2
##    species          bmi
##    <chr>          <dbl>
##  1 Hutt           443.
##  2 Vulptereen      50.9
##  3 Yoda's species  39.0
##  4 Kaleesh         34.1
##  5 Droid           32.7
##  6 Dug             31.9
##  7 Trandoshan      31.3
##  8 Sullustan       26.6
##  9 Zabrak          26.1
## 10 Besalisk        26.0
## # i 22 more rows
```

1.10 Exercises

1. LogSumExp trick

Try to implement the log-sum-exp trick in a function that takes as argument three `numeric` variables and computed the log of the sum of the exponentials in a numerically stable way. See this Wiki paragraph if you don't know the trick yet.

```
log_sum_exp3 <- function(a, b, c) {
  # delete this function and re-write it using the trick
  # in order to make it work
  return(log(exp(a) + exp(b) + exp(c)))
}

# test - this result is obviously wrong: edit the function above
log_sum_exp3(a = -1000, b = -1001, c = -999) # should give -998.5924
```

```
## [1] -Inf
```

Chapter 2

Some elementary statistics problems

In the following paragraphs, we will see some applications of the R software to solve elementary statistics problems.

2.1 Hypothesis testing

2.1.1 Example

The rector of Politecnico di Torino wants to monitor the spread of Covid-19 virus inside the university, to check whether it is in line with the overall diffusion on the Italian territory, or whether it is higher, and therefore requires extra safety measures.

Tests are made on a sample of $n = 250$ students out of the total $N = 5000$ students (because of whatever reasons, such as economic or sustainability concerns). We assume that the tests are *perfect*, meaning that they have 100% sensitivity (true positive rate) and specificity (true negative rate).

We know that $p_0 = 0.015$ is the prevalence of the virus in the Italian population at the end of 2021, and we are given a *warning* threshold for the prevalence of $p_1 = 0.04$, above which the rector will have reasons to adopt more restrictive measures.

2.1.2 False alarm

> **Question**: given a threshold x_0 of positive tests, what are the false alarm and the false non-alarm probabilities?

Formally, a false alarm (type I error) probability is defined as the probability of the r.v. counting the positive tests X being greater than or equal to the threshold x_0, conditioned on the fact that the prevalence is not different than the Italian reference value p_0. I.e.

$$P(X \geq x_0 | p = p_0)$$

Recall that $X \sim \text{Hypergeometric}(N, K, n)$ where $p = K/N$.

The pmf is the following

$$P(X = x) = \frac{\binom{K}{x}\binom{N-K}{n-x}}{\binom{N}{n}},$$

which leads to the false alarm probability computation

$$P(X \geq x_0 | p = p_0) = \sum_{k=x_0}^{n} P(X = k) = \sum_{k=x_0}^{n} \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}.$$

When N is large enough and p small enough, we can use the Binomial approximation. To be clear, with such approximation, we assume that while sampling from the population, the proportion of positives stays the same (which in our case it is safe to assume).

Therefore we approximate the false alarm probability with

$$P(X \geq x_0 | p = p_0) = \alpha \approx \sum_{k=x_0}^{n} \binom{n}{k} p^k (1-p)^{n-k}.$$

Similarly, we compute the false non-alarm probability (type II error), defined as the probability of the r.v. counting the positive tests X being lower than the threshold x_0, conditioned on the fact that the prevalence is the one we identified as worrying, i.e. $p = p_1$.

$$P(X < x_0 | p = p_1) = \beta \approx \sum_{k=0}^{x_0 - 1} \binom{n}{k} p^k (1-p)^{n-k}.$$

To answer the question, let's now compute these quantities using the R functions `phyper` and `pbinom`. We can do this for several values of x_0 i.e. $\forall x_0 \in \{1, \dots, 10\}$ so to understand how α and β change.

```r
# import the main libraries
library(tidyverse)
```

```r
# set random seed for reproducibility
set.seed(42)

# define the parameters of the test
N <- 5000
n <- 250
p0 <- .015
p1 <- .04

# get the lower integer part in case the product is not natural
K0 <- floor(N * p0)
K1 <- floor(N * p1)

# false positive (check phyper params with ?phyper)
# we subtract 1 to x0 because we are looking for P(X >= x0) = 1 - P(X < x0-1),
# not P(X > x0)
fp_df <- tibble(
  x0 = 1:10,
  hyp = 1 - phyper(x0 - 1, K0, N - K0, n),
  bin = 1 - pbinom(x0 - 1, n, p0)
)
# let's view the result
fp_df
```

```
## # A tibble: 10 x 3
##       x0     hyp     bin
##    <int>   <dbl>   <dbl>
##  1     1 0.979   0.977
##  2     2 0.896   0.890
##  3     3 0.732   0.725
##  4     4 0.521   0.517
##  5     5 0.321   0.322
##  6     6 0.171   0.176
##  7     7 0.0795  0.0847
##  8     8 0.0325  0.0364
##  9     9 0.0118  0.0141
## 10    10 0.00382 0.00494
```

```r
# false negative
# again, we subtract 1: P(X < x0) = P(X <= x0-1) and
# pbinom(x, n, p) := P(X <= x; n, p)
fn_df <- tibble(
  x0 = 1:10,
  hyp = phyper(x0 - 1, K1, N - K1, n),
  bin = pbinom(x0 - 1, n, p1)
)
```

```
fn_df
```

```
## # A tibble: 10 x 3
##       x0       hyp       bin
##    <int>     <dbl>     <dbl>
##  1     1 0.0000283 0.0000370
##  2     2 0.000339  0.000422
##  3     3 0.00203   0.00242
##  4     4 0.00810   0.00930
##  5     5 0.0243    0.0270
##  6     6 0.0587    0.0633
##  7     7 0.119     0.125
##  8     8 0.208     0.215
##  9     9 0.322     0.328
## 10    10 0.452     0.455
```

We observe that with $x_0 = 7$ we have

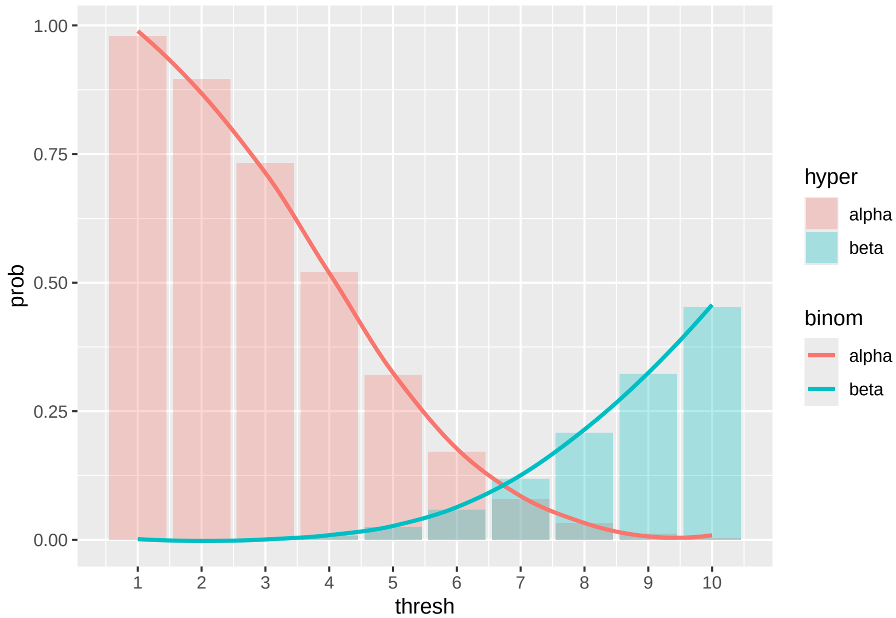
$$\alpha = 0.0795, \beta = 0.119$$

which jointly optimize the two probabilities. In order to get a more complete overview, we could plot those values together

```
library(scales) # to define custom scale on axes
```

```
# join the two dataframes
tot_df <- left_join(fp_df, fn_df,
  by = "x0", suffix = c("alpha", "beta")
)
tot_df %>%
  ggplot() +
  geom_bar(aes(x0, hypalpha, fill = "alpha"), alpha = .3, stat = "identity") +
  geom_bar(aes(x0, hypbeta, fill = "beta"), alpha = .3, stat = "identity") +
  geom_smooth(aes(x0, binalpha, color = "alpha"), se = FALSE) +
  geom_smooth(aes(x0, binbeta, color = "beta"), se = FALSE) +
  labs(x = "thresh", y = "prob", fill = "hyper", color = "binom") +
  # print integers on the x axis
  scale_x_continuous(breaks = pretty_breaks(10))
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



The plot indeed suggests that $x_0 = 7$ is a good compromise, but there is no rule in choosing such threshold, and in fact, we might be more interested in a safer threshold in terms of lower false alarm probability, or lower false non-alarm probability.

This decision process can be thought as two simple hypotheses test: we consider the null hypothesis $H_0 : p = p_0$. Given our observation, i.e. x, count of positive tests, realization of the X r.v.

- if $x < x_0$ we have no reasons to reject H_0
- if $x \geq x_0$ we reject H_0 and accept the alternative hypothesis

2.1.3 The Binomial approximation

Exercise: The plot above shows that the Binomial distribution offers indeed a good approximation of the Hypergeometric. However, as an exercise, we can repeat the experiment changing the values of N and p.

> repeat the computations above tweaking the parameters and observe the resulting plot. With which parameters do you observe a poor approximation of the Hypergeometric with the Binomial distribution? Feel free to plot different quantities, such as the density instead of the cumulative distribution

2.2 Confidence intervals

In case we have two samples $\mathbf{X} = X_1, ..., X_m$ and $\mathbf{Y} = Y_1, ..., Y_n$ coming from two unknown Normal distributions, we might want to test whether they come from distributions with same mean, or in other words, whether they are both scattered around a common value. For example, we could be interested in checking whether a set of jewelry items, which have some variability in weight, has been produced in the same original factory or if it is counterfeit.

After defining a confidence level $1 - \alpha$ and a variable that we want to limit with some "confidence", we can compute the confidence interval (Δ_l, Δ_u). The confidence interval tells us where the real value of the variable of interest can fall, with some confidence given the evidence. The higher is the confidence that we want to enforce, the larger the interval will be, but remember that a wide confidence interval is often not useful.

2.2.1 Example

Let's start with a simple example, where we have two normal samples, of which we know nothing (not the mean, nor the variance):

$$X_1, ..., X_m \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu_x, \sigma_x^2)\,, \quad Y_1, ..., Y_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu_y, \sigma_y^2)$$

The goal is to get a confidence interval for the variable $\mu_x - \mu_y$, therefore an interval in which we can expect the two means difference to be with confidence $1 - \alpha$. Assuming that both n, m are sufficiently large, the CI is computed as

$$\bar{\mathbf{x}} - \bar{\mathbf{y}} \pm z_{\alpha/2}\sqrt{\frac{s_x^2}{m} + \frac{s_y^2}{n}}\,.$$

Let's simulate some data and use R to get the confidence interval, using as paramters $\mu_x = 10, \sigma_x^2 = 36$ and $\mu_y = 10, \sigma_y^2 = 49$. We draw $m = 100$ and $n = 120$ samples respectively.

```r
# set the parameters
m <- 100
mu_x <- 10
sigma_x <- 6
n <- 120
mu_y <- 10
sigma_y <- 7

# use rnorm to draw independent and identically distributed samples
x <- rnorm(m, mu_x, sigma_x)
y <- rnorm(n, mu_y, sigma_y)
```

At this point we can define the confidence level and compute the CI.

```
alpha <- 0.05 # typical value for alpha

# define a custom function based on the formula above
confidence_interval <- function(x, y, alpha) {
  # note that mean() is the sample mean
  dev <- mean(x) - mean(y)
  # var() is the sample variance, not the actual variance
  delta <- qnorm(1 - alpha / 2) * sqrt(var(x) / m + var(y) / n)
  return(c(dev - delta, dev + delta))
}

confidence_interval(x, y, alpha)
```

```
## [1] -0.859210  2.540521
```

We notice that the interval contains the value 0, and this is due to the fact that both samples have same mean, therefore the sample means will also be, with enough samples, very similar, thus with zero (or close to zero) difference.

To wrap this up, under large sample assumptions and unknown variance, we computed the confidence interval of a two-samples difference. More in general, when we compare two samples $\mathbf{x}, \mathbf{y}$ there are multiple CIs that we might have to compute. We go through them below.

2.2.2 The four cases

To better clarify, we can divide the confidence intervals over two samples in four cases:

1. we know the variances of the two samples, σ_x^2, σ_y^2,
2. we don't know the variances, but the samples sizes are large enough,
3. we don't know the variances and we cannot assume large sample size, but we assume homoscedasticity (equal variances),
4. we don't know the variances, samples size is not large and homoscedasticity cannot be assumed.

2.2.2.1 Case 1

The variable of interest is, just like in any of the following cases, $\mu_x - \mu_y$, which we don't know.

What we do know, given normality of the samples and from some background theory, is that

$$\overline{\mathbf{X}} - \overline{\mathbf{Y}} \sim \mathcal{N}(\mu_x - \mu_y, \frac{\sigma_x^2}{m} + \frac{\sigma_y^2}{n}) .$$

Normalizing, we get

$$\frac{\overline{\mathbf{X}} - \overline{\mathbf{Y}} - (\mu_x - \mu_y)}{\sqrt{\frac{\sigma_x^2}{m} + \frac{\sigma_y^2}{n}}} \sim \mathcal{N}(0,1)\,,$$

and thus, to get a confidence interval, we simply write the *coverage probability*

$$P(-z_{\alpha/2} < \frac{\overline{\mathbf{X}} - \overline{\mathbf{Y}} - (\mu_x - \mu_y)}{\sqrt{\frac{\sigma_x^2}{m} + \frac{\sigma_y^2}{n}}} < z_{\alpha/2}; \mu_x, \mu_y) \equiv 1 - \alpha$$

and rearranging the terms we find the confidence interval for $\mu_x - \mu_y$ in case 1:

$$\left(\overline{\mathbf{X}} - \overline{\mathbf{Y}} - z_{\alpha/2}\sqrt{\frac{\sigma_x^2}{m} + \frac{\sigma_y^2}{n}}, \overline{\mathbf{X}} - \overline{\mathbf{Y}} + z_{\alpha/2}\sqrt{\frac{\sigma_x^2}{m} + \frac{\sigma_y^2}{n}}\right)$$

2.2.2.2 Case 2

If m, n are "large", then the sample variance will asymptotically get closer to the variance. In the previous example we assumed large sample size, that's why we simply replaced σ with s in the above formula.

The interval

$$\bar{\mathbf{x}} - \bar{\mathbf{y}} \pm z_{\alpha/2}\sqrt{\frac{s_x^2}{m} + \frac{s_y^2}{n}}\,.$$

is a confidence interval for $\mu_x - \mu_y$ with *approximately* level $1 - \alpha$.

2.2.2.3 Case 3

In case we assume homoscedasticity, we leverage on the observation as much as we can, therefore we compute a weighted average (*pooled*) of the sample variances and we use that estimator to approximate the common variance.

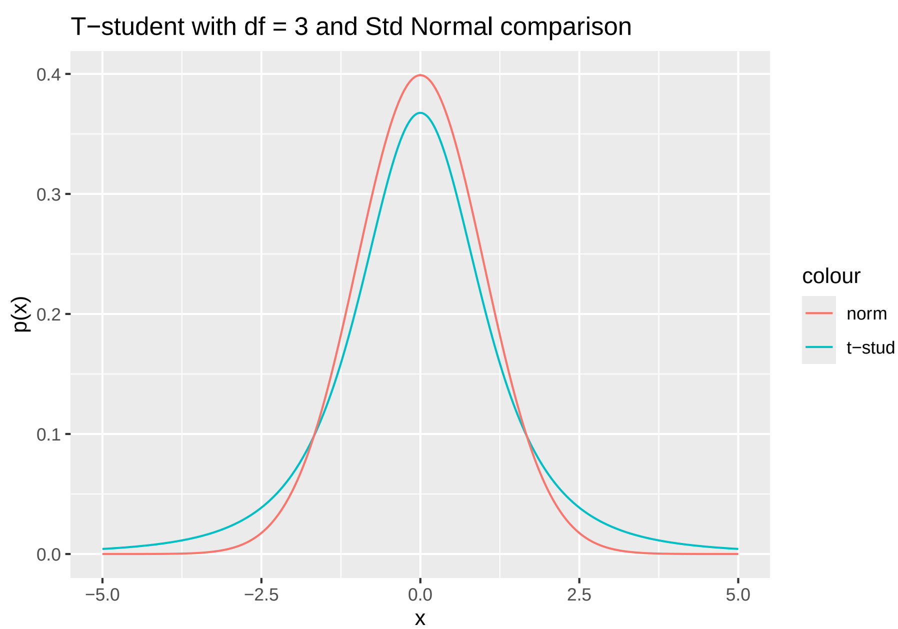
$$s_p^2 = \frac{(m-1)s_x^2 + (n-1)s_y^2}{m+n-2}\,.$$

One can prove that the r.v $\frac{(m+n-2)s_p^2}{\sigma^2} \sim \chi^2(m+n-2)$ and from that follows

$$\frac{\overline{\mathbf{X}} - \overline{\mathbf{Y}} - (\mu_x - \mu_y)}{\sqrt{s_p^2\left(\frac{1}{m} + \frac{1}{n}\right)}} \sim t(m+n-2)\,,$$

Where $t(d)$ is a T-student distribution with d degrees of freedom and, compared to a standard Normal distribution, has larger tails, which means that the quantiles over small values such as α will be higher wrt the std normal.

```
x_dt <- seq(-5, 5, by = 0.01)
norm_t_df <- tibble(x = x_dt, p_dt = dt(x, df = 3), p_nt = dnorm(x))

norm_t_df %>%
  ggplot() +
  geom_line(aes(x, p_dt, color = "t-stud")) +
  geom_line(aes(x, p_nt, color = "norm")) +
  labs(
    title = "T-student with df = 3 and Std Normal comparison",
    x = "x", y = "p(x)"
  )
```



Following the same approach of the previous cases, the confidence interval is found using the quantiles of the specific T-student distributions.

Before we computed the CI “manually”, but luckily there exists a function which already implements this formula (and not only this):

```
# set var.equal = TRUE for pooled variance
test_obj <- t.test(x, y, conf.level = 1 - alpha, var.equal = TRUE)
test_obj # outputs some information
```

```
##
##  Two Sample t-test
##
## data:  x and y
```

```
## t = 0.9646, df = 218, p-value = 0.3358
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.8770073  2.5583183
## sample estimates:
## mean of x mean of y
## 10.195089  9.354433
```

```
test_obj$conf.int # we're interested in the confidence interval
```

```
## [1] -0.8770073  2.5583183
## attr(,"conf.level")
## [1] 0.95
```

2.2.2.4 Case 4

Otherwise, in case we cannot assume that the two samples have the same variance. In R this specific confidence interval can be computed setting the `var.equal` flag to false in `t.test()`.

```
# or leaving it to default = F
t.test(x, y, var.equal = FALSE) # conf.level = 0.95 by default
```

```
##
##  Welch Two Sample t-test
##
## data:  x and y
## t = 0.96929, df = 214.36, p-value = 0.3335
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.8688616  2.5501726
## sample estimates:
## mean of x mean of y
## 10.195089  9.354433
```

This is called the Welch-Satterthwaite approximation.

2.3 P-value

In the context of hypothesis testing, the *p-value* is the probability, under the null hypothesis, that we obtain a test statistic at least as contradictory to H_0 as the statistic from the available sample. In other words, it's a measure of the null hypothesis support in a 0 to 1 range.

2.3.1 Example

In a fair French roulette, the red probability is 18/37.

Questions:

a. we observe 20 reds out of 30. Compute the p-value of the null hypothesis of the roulette being fair, with respect to the alternative hypothesis of the roulette being skewed towards the red,
b. in the same situation, compute the p-value of the null hypothesis of the roulette being fair wrt the alternative hypothesis of it not being fair,
c. repeat a. and b. with 200 reds out of 380 as outcome.

Answers:

a. Given $X = \#$reds, we can compute the following probability under the null hypothesis, noticing that $X \sim \text{Binomial}(n, p)$ with $n = 30$:

$$P(X \geq 20|H_0) = \sum_{k=20}^{30} \binom{30}{k} \left(\frac{18}{37}\right)^k \left(\frac{19}{37}\right)^{(30-k)}$$

Note that we specify the condition ≥ 20 as the p-value stands for the probability of observing an outcome **at least** as contradictory as the one observed.

In R, we can compute it "by hand":

```
p0 <- 18 / 37
n <- 30
# P(X > 19) \equiv P(X >= 20)
pbinom(19, n, p0, lower.tail = FALSE) # notice the lower.tail flag
```

```
## [1] 0.03598703
```

or use the Binomial test function

```
# check how it works
?binom.test
```

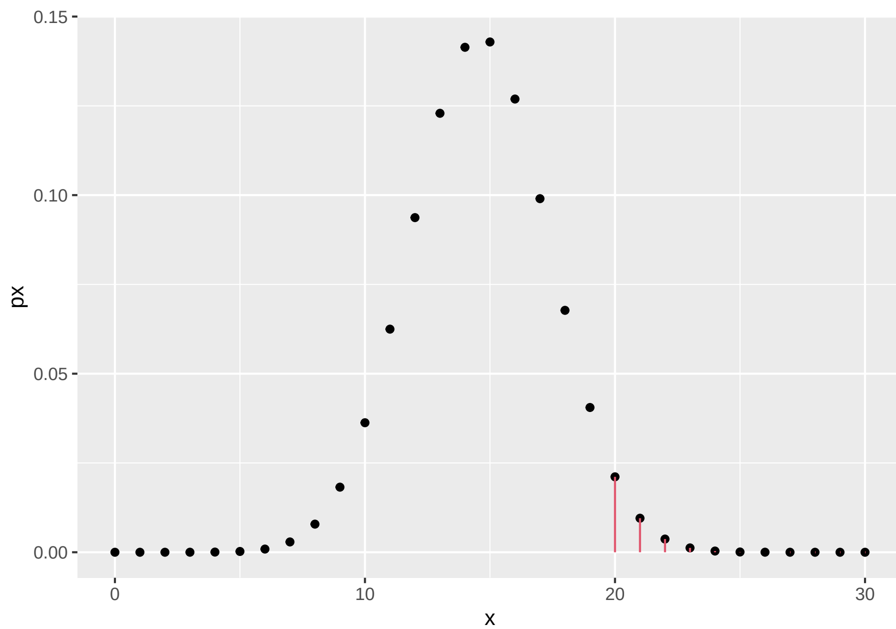
```
bin_test <- binom.test(20, n, p0, alternative = "greater")
bin_test
```

```
##
##  Exact binomial test
##
## data:  20 and n
## number of successes = 20, number of trials = 30, p-value = 0.03599
## alternative hypothesis: true probability of success is greater than 0.4864865
## 95 percent confidence interval:
##  0.5005613 1.0000000
## sample estimates:
## probability of success
##              0.6666667
```

```
bin_test$p.value
```

```
## [1] 0.03598703
```

```
# sketch of "greater" type p-value
sketch_df <- tibble(x = 0:n, px = dbinom(x, n, p0))
sketch_df %>%
  ggplot() +
  geom_point(aes(x, px)) +
  geom_segment(
    data = . %>% dplyr::filter(x >= 20),
    aes(x = x, xend = x, y = 0, yend = px), color = 2
  )
```



b. If the alternative hypothesis is just $H_1 : p \neq 18/37$ then the contradiction can happen both on the left and on the right side of the curve, i.e. the roulette favors the red or the roulette favors the black *in the same or more extreme way*. Therefore we have to sum the two probabilities:

$$P(X \geq 20|H_0) + P(X < 10|H_0)\,.$$

Note: where does the 10 come from? Well, if we observe 20 reds out of 30, at first we might think that the roulette prefers red instead of black. One might argue that it does not prefer red, but rather it is just not fair (no matter if it favors red or black). We then have to consider *all* cases that happen with a lower or equal probability than the sample's. If the distribution is symmetric (and in this case it almost is), we just consider $X < 30 - 20$. Check the probability $P(X = 20)$ and find which values on the left tail gives at most the same probability. Those are the values that must be considered in the two-sided test.

We compute this quantity first using `pbinom` as before

```
pbinom(19, n, p0, lower.tail = FALSE) + pbinom(9, n, p0)
```

```
## [1] 0.06614782
```

and then using the test function

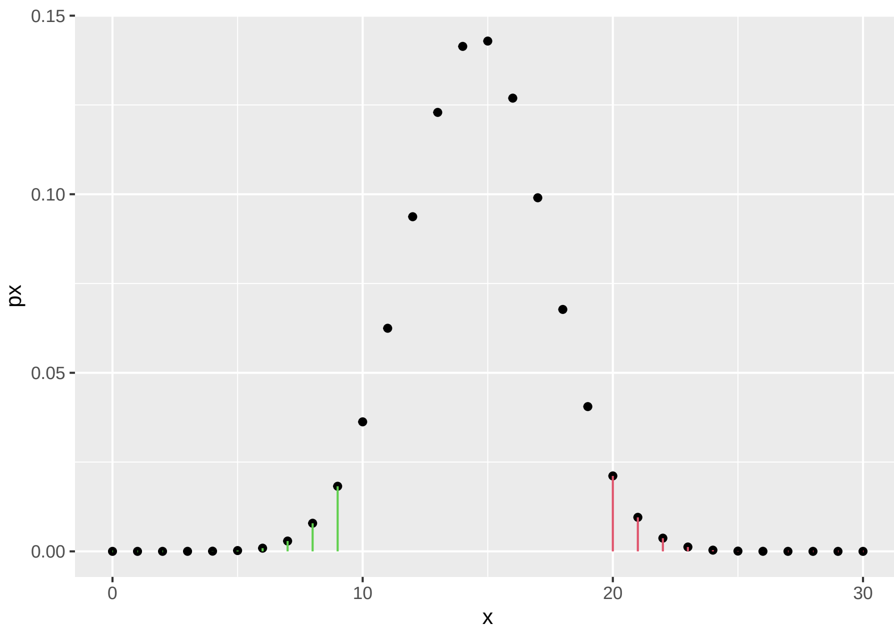
```
bin_test_2 <- binom.test(20, n, p0, alternative = "two.sided")

bin_test_2
```

```
##
##  Exact binomial test
##
## data:  20 and n
## number of successes = 20, number of trials = 30, p-value = 0.06615
## alternative hypothesis: true probability of success is not equal to 0.4864865
## 95 percent confidence interval:
##  0.4718800 0.8271258
## sample estimates:
## probability of success
##              0.6666667
```

```
bin_test_2$p.value
```

```
## [1] 0.06614782
```

```
# sketch of bilateral type p-value
sketch_df <- tibble(x = 0:n, px = dbinom(x, n, p0))
sketch_df %>%
  ggplot() +
  geom_point(aes(x, px)) +
  geom_segment(
    data = . %>% dplyr::filter(x >= 20),
    aes(x = x, xend = x, y = 0, yend = px), color = 2
  ) +
  geom_segment(
    data = . %>% dplyr::filter(x < 10),
    aes(x = x, xend = x, y = 0, yend = px), color = 3
  )
```



c. When n increases, and we can consider it large enough, we can use the normal approximation of the binomial (for the CLT), i.e.

$$Z = \frac{X - np}{\sqrt{np(1-p)}} \overset{n \to +\infty}{\approx} \mathcal{N}(0,1)\,.$$

Under the null hypothesis, in the unilateral case, we compute the p-value as

$$P(Z \geq z | H_0) = 1 - \Phi(z)\,,$$

where z is the realization of Z given that the realization of X is $x = 200$ (considered to be large enough for the approximation).

In R:

```
n <- 380
x <- 200
z <- (x - n * p0) / sqrt(n * p0 * (1 - p0))
pnorm(z, lower.tail = FALSE)
```

```
## [1] 0.06016385
```

or, using the R proportion test `prop.test`

```
?prop.test
```

```
prop.test(x, n, p = p0, alternative = "greater") # uses continuity correction
```

```
##
##  1-sample proportions test with continuity correction
##
## data:  x out of n, null probability p0
## X-squared = 2.2562, df = 1, p-value = 0.06654
## alternative hypothesis: true p is greater than 0.4864865
## 95 percent confidence interval:
##  0.4828353 1.0000000
## sample estimates:
##         p
## 0.5263158
```

Notice that the result is not exactly the same. This is due to the fact that we are approximating a discrete probability sum with an integral of a Gaussian. To correct for this, it's enough to subtract (or add, depending on the case) `.5` to the x value.

```
z_corr <- (x - .5 - n * p0) / sqrt(n * p0 * (1 - p0))
pnorm(z_corr, lower.tail = FALSE)
```

```
## [1] 0.06653798
```

> The bilateral case with Normal approximation is left to the reader as exercise

Chapter 3

Normal sampling

3.1 Random and non-random normal samples

First we look at the simplest case: independent and identically distributed Normal random variables.

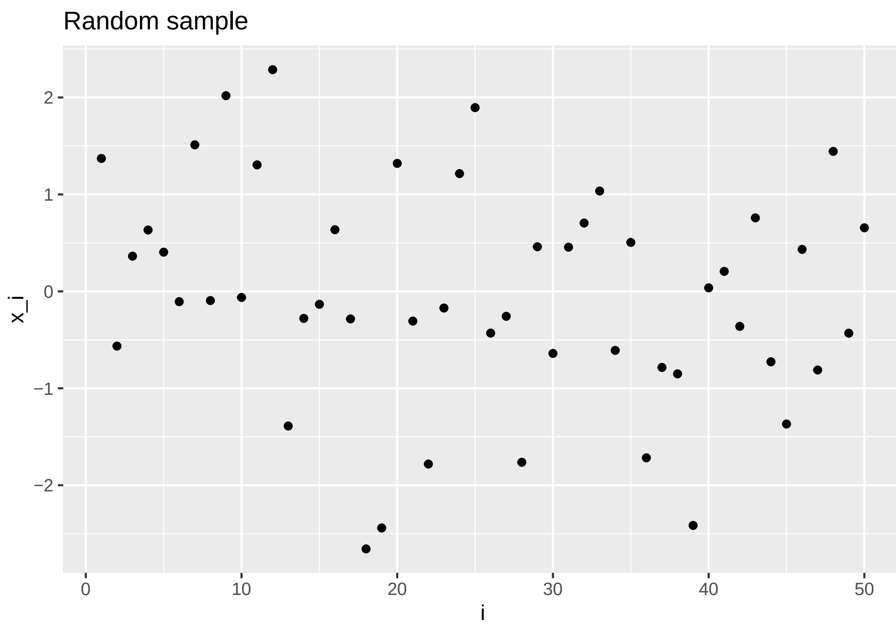
$$\mathbf{X} \sim \mathcal{N}\left(\begin{pmatrix}\mu \\ \vdots \\ \mu\end{pmatrix}, \begin{pmatrix}\sigma^2 & & 0 \\ & \ddots & \\ 0 & & \sigma^2\end{pmatrix}\right)$$

In order to simulate a sample of i.i.d. Normal r.v., we need to call the `rnorm` function, which uses R pseudo-random number generator. Like every pseudo-RNG, it allows for specifying a seed, that is useful for reproducibility purposes (see pseudo-RNG wiki for more details).

This is how you can simulate a sample with $n = 50$ i.i.d. Normal random variables in R.

```
set.seed(42) # for reproducibility
library(ggplot2) # plotting
library(dplyr) # dataframe manipulation
library(tibble) # tibble

n <- 50
rnorm_sample <- rnorm(n) # mu = 0, sigma = 1, for instance

iidplt <- rnorm_sample %>%
  enframe() %>% # creates tibble with name,value columns
  ggplot() +
```

```
  geom_point(aes(x = name, y = value)) +
  labs(title = "Random sample") + # add title and labels to the plot
  xlab("i") +
  ylab("x_i")
iidplt
```



If the random variables are independent but distributed with different mean, we can identify two notable cases: **mean shift** and **mean drift**.

In the first case, we write

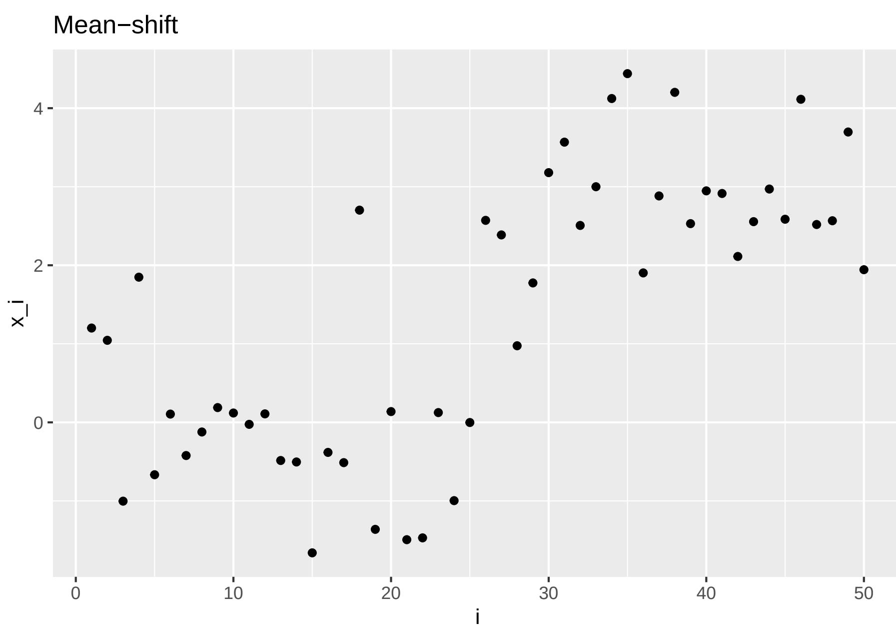
Mean shift:

$$\mathbf{X} \sim \mathcal{N}\left(\begin{pmatrix}\mu_0\\ \vdots\\ \mu_0\\ \mu_1\\ \vdots\\ \mu_1\end{pmatrix}, \begin{pmatrix}\sigma^2 & & 0\\ & \ddots & \\ 0 & & \sigma^2\end{pmatrix}\right),$$

and in R we can simulate such sample as follows

```
# (optional) creates a function to simplify other plots
plot_sample <- function(x, title = NULL, xlab = NULL, ylab = NULL) {
  plt <- x %>%
    enframe() %>%
    ggplot(aes(x = name, y = value)) +
    geom_point() +
```

```
    labs(title = title) +
    xlab(xlab) +
    ylab(ylab)
  return(plt)
}

# first half with mean = 0, second half with mean = 3
# simulate by concatenating two rnorm samples
ms_sample <- c(rnorm(floor(n / 2)), rnorm(n - floor(n / 2), 3))
# or equivalently, by concatenating two mean vectors in one rnorm call
ms_sample <- rnorm(n, mean = c(
  rep(0, floor(n / 2)),
  rep(3, n - floor(n / 2))
))

# save the plot in a variable for later use
msplt <- plot_sample(ms_sample, "Mean-shift", "i", "x_i")
msplt
```



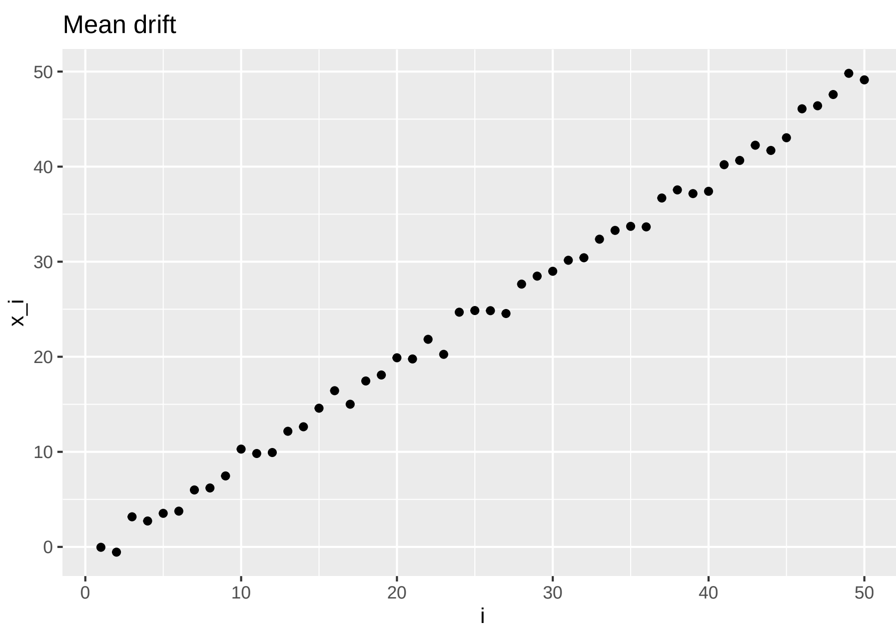
For the mean drift, the mean changes variable by variable

Mean drift:

$$\mathbf{X} \sim \mathcal{N}\left(\begin{pmatrix}\mu_1\\ \mu_2\\ \vdots\\ \mu_n\end{pmatrix}, \begin{pmatrix}\sigma^2 & & 0\\ & \ddots & \\ 0 & & \sigma^2\end{pmatrix}\right),$$

Similarly, in R, we can simulate a mean drift with mean going from 0 to $n-1$ with unitary step.

```r
# mean is a range vector
md_sample <- rnorm(n, 0:(n - 1))

mdplt <- plot_sample(md_sample, "Mean drift", "i", "x_i")
mdplt
```



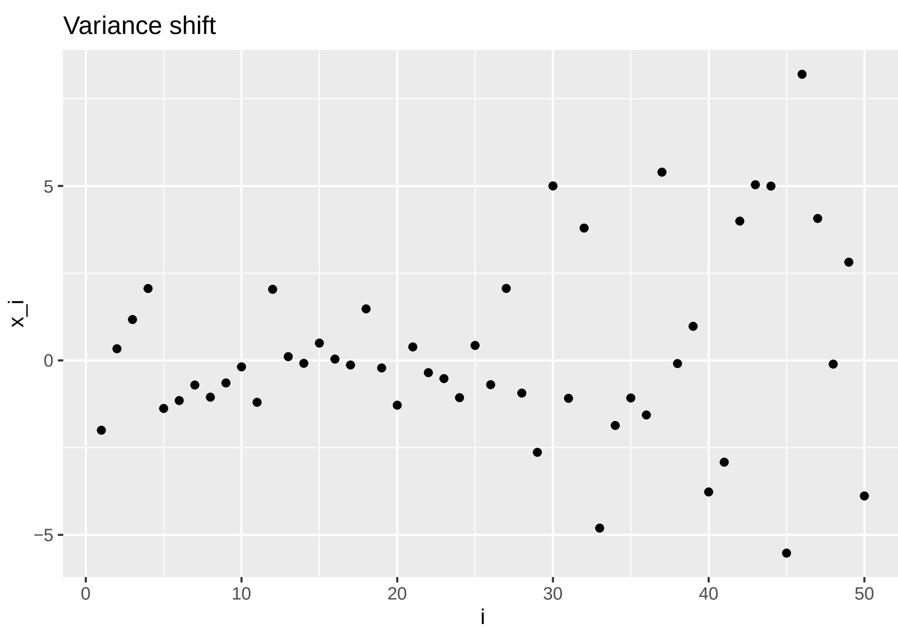
The same concept is applied also to random variables drawn by a normal with changing variance.

Variance shift:

$$\mathbf{X} \sim \mathcal{N}\left(\begin{pmatrix}\mu\\ \vdots\\ \mu\end{pmatrix}, \begin{pmatrix}\sigma_0^2 & & & & 0\\ & \ddots & & & \\ & & \sigma_0^2 & & \\ & & & \sigma_1^2 & \\ & 0 & & & \ddots & \\ & & & & & \sigma_1^2\end{pmatrix}\right),$$

in R:

```
vs_sample <- c(
  rnorm(floor(n / 2)), # first half
  rnorm(n - floor(n / 2), sd = 4)
) # second half

vsplt <- plot_sample(vs_sample, "Variance shift", "i", "x_i")
vsplt
```

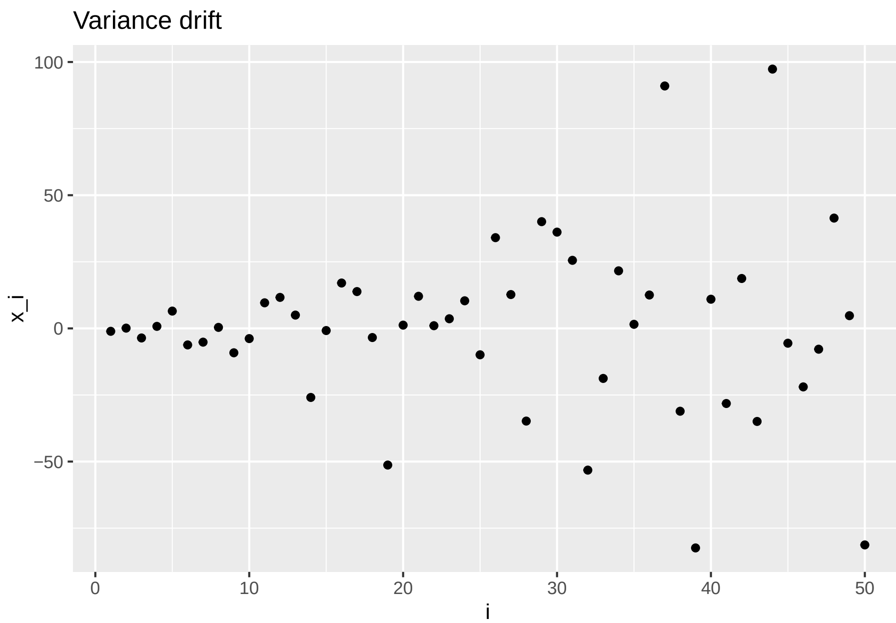


Variance drift

$$\mathbf{X} \sim \mathcal{N}\left(\begin{pmatrix}\mu \\ \vdots \\ \mu\end{pmatrix}, \begin{pmatrix}\sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_n^2\end{pmatrix}\right),$$

in R:

```
vd_sample <- rnorm(n, sd = 1:n)

vdplt <- plot_sample(vd_sample, "Variance drift", "i", "x_i")
vdplt
```



3.1.1 Autocorrelated sample

Let $\mathbf{Y} = Y_1, ..., Y_{n+1}$ an i.i.d. standard Normal random sample (i.e. $Y_i \sim \mathcal{N}(0, 1)$).

Then $\mathbf{X} = X_1, ..., X_n$ such that $X_i = Y_{i+1} - Y_i$ is a Normal non-random sample with such distribution

$$\mathbf{X} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & & 0 & \\ & & & & \\ & \ddots & \ddots & \ddots & \\ & & & & \\ & 0 & & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}\right).$$

The proof is left as exercise to the reader.

> Hint: first compute $E[X_i]$, then $\text{Var}(X_i)$ and finally $\text{Cov}(X_{i+1}, X_i)$

In R, we build this simulation by simulating $\mathbf{Y}$ and deriving $\mathbf{X}$

```
y <- rnorm(n)
# index by removing first and last elements
auto_sample <- y[-1] - y[-n]
```

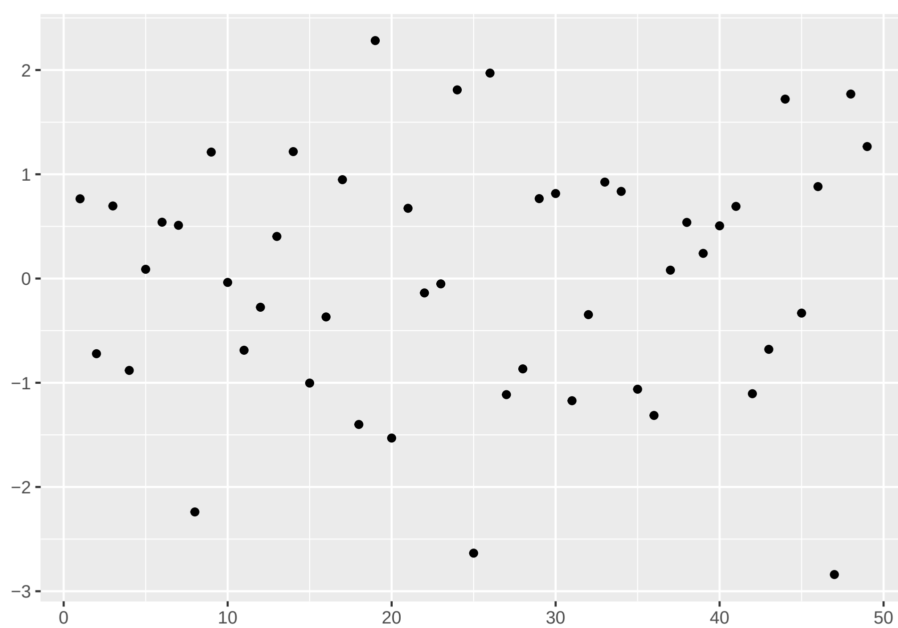
```
# equivalent to
auto_sample <- y[2:n] - y[1:(n - 1)]
```

or, simpler

```
auto_sample <- y %>%
  diff()
auto_sample
```

```
##  [1]  0.76486294 -0.72125125  0.69608123 -0.88154477  0.08858529  0.54025680
##  [7]  0.51057417 -2.23933913  1.21290797 -0.03747466 -0.68795847 -0.27543871
## [13]  0.40362741  1.21753967 -1.00313038 -0.36856515  0.94830413 -1.40013055
## [19]  2.28258801 -1.53046919  0.67372423 -0.13812248 -0.05124700  1.80962271
## [25] -2.63471000  1.97079841 -1.11389726 -0.86677899  0.76644216  0.81553534
## [31] -1.17229042 -0.34656350  0.92484634  0.83583274 -1.06205518 -1.31356822
## [37]  0.08035065  0.53828058  0.24128720  0.50568106  0.69192514 -1.10511511
## [43] -0.67926274  1.72098578 -0.33164903  0.88189010 -2.83982793  1.76979995
## [49]  1.26604281
```

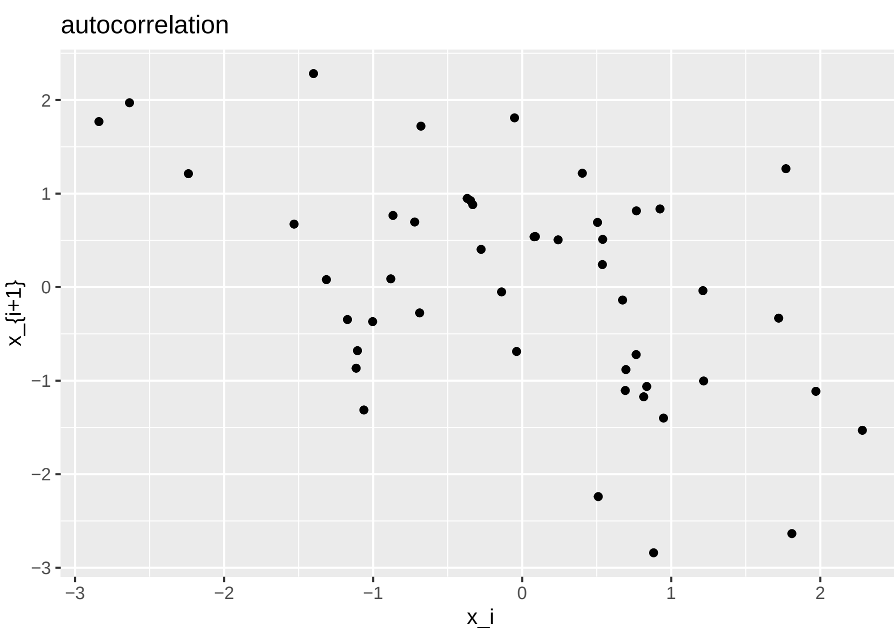
```
plot_sample(auto_sample)
```



From this plot it is almost impossible to spot the correlation between variables. It's more visible with a plot having X_i on one axis and X_{i+1} on the other.

```
autoplt <- auto_sample %>%
  { # group in curly brackets so that tibble is called
    # without passing auto_sample as first implicit argument
```

```
    tibble(x = .[-length(.)], y = .[-1]) # dot as placeholder
    # equivalent (but more efficient) to:
    # tibble(x = auto_sample[-length(auto_sample)], y = auto_sample[-1])
  } %>%
  ggplot() +
  geom_point(aes(x, y)) +
  labs(title = "autocorrelation") +
  xlab("x_i") +
  ylab("x_{i+1}")
autoplt
```



Maybe it's even more clear if the number of observations increases. Try with larger n.

Questions:

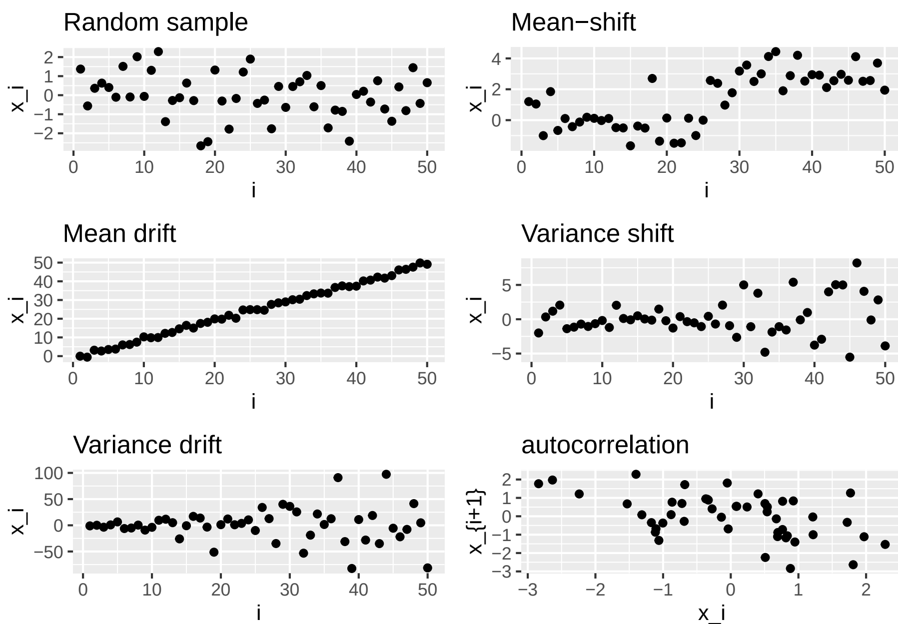
- how is X_i distributed?
- what is the correlation between two subsequent samples i.e. $\rho(X_{i+1}, X_i)$

Exercise: build `Xpos` such that correlation is positive

We can compare these models by grouping all the plots in one single picture.

```
# install.packages("ggpubr")
library(ggpubr)
ggarrange(iidplt, msplt, mdplt, vsplt, vdplt, autoplt,
  nrow = 3, ncol = 2
```

```
)
```



3.2 Exchangeable normal random variables

Let $\mathbf{X} = (X_1, ..., X_n)$ be a random vector whose elements follow the conditional distribution

$$X_i | \mu \sim \mathcal{N}(\mu, \sigma^2)$$

with μ being another normally distributed r.v.

$$\mu \sim \mathcal{N}(\mu_0, \sigma_0^2)\,.$$

X_i are conditionally independent given μ, and also, their joint distribution is equal for any permutation of the random vector elements (exchangeable). But if we don't know μ, what is the distribution of X_i (marginal distribution)?

It's

$$\mathbf{X} \sim \mathcal{N}\left(\begin{pmatrix}\mu_0 \\ \vdots \\ \mu_0\end{pmatrix}, \begin{pmatrix}\sigma^2 + \sigma_0^2 & & & & \\ & \ddots & & \sigma_0^2 & \\ & & \ddots & & \\ & \sigma_0^2 & & \ddots & \\ & & & & \sigma^2 + \sigma_0^2\end{pmatrix}\right)$$

Exercise: prove it (i.e. find expected value, variance and covariance).

Hint: $E[X_i] = E_{f\sim\mu}[E_{f\sim X_i|\mu}[X_i|\mu]]$

In R, we simply simulate a realization of the mean r.v., then use that value to simulate the $\mathbf{X}$ Normal sample given μ.

```
mu <- rnorm(1)
x <- rnorm(n, mu)
```

3.3 Multivariate Normal random samples

Multivariate normal distribution functions are provided by the `mvtnorm` library.

Take this as an example:

$$\begin{pmatrix} X \\ S \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 3 \end{pmatrix}\right)$$

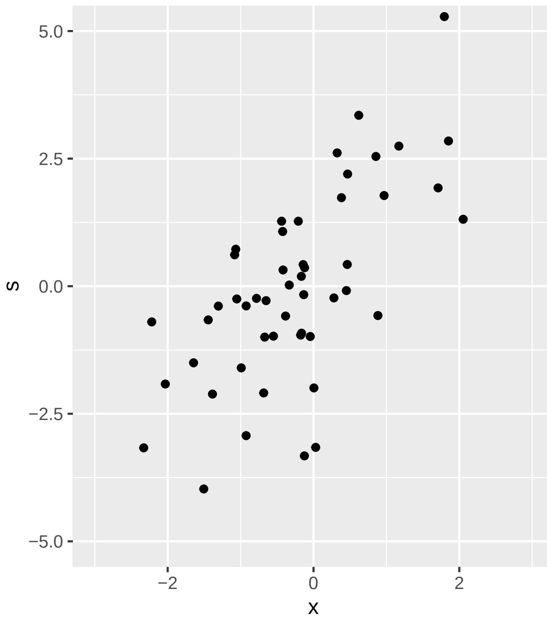
```
# install.packages("mvtnorm")
library(mvtnorm)

# simulate n bivariate normal r.v.
m <- rep(0, 2) # mean vector
vcov_mat <- matrix(c(1, 1, 1, 3), nrow = 2) # Sigma

# specify mean vector and var-cov matrix
bvt_samples <- mvtnorm::rmvnorm(n, m, vcov_mat)
head(bvt_samples, 10) # print only the first 10 elements
```

```
##              [,1]       [,2]
##  [1,] -0.78186845 -0.2389248
##  [2,] -0.17599524 -0.9582082
##  [3,] -0.92364519 -0.3863103
##  [4,]  1.17077708  2.7456040
##  [5,] -2.03263913 -1.9171945
##  [6,] -0.92418665 -2.9286612
##  [7,] -0.99088212 -1.6003554
##  [8,] -1.50519809 -3.9739964
##  [9,] -0.04465341 -0.9858018
## [10,]  1.70934985  1.9264325
```

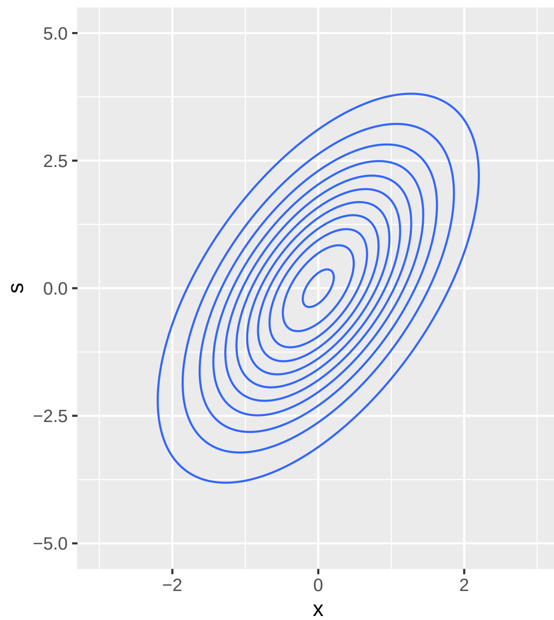
Let's plot this sample

```
bvt_samples_df <- tibble(x = bvt_samples[, 1], s = bvt_samples[, 2])
bvt_scatter <- bvt_samples_df %>%
  ggplot() +
  geom_point(aes(x, s)) +
  coord_fixed(xlim = c(-3, 3), ylim = c(-5, 5), ratio = -.7)
```

```
bvt_scatter
```




```
# plot the true distribution
# generate a grid (many points with fixed space between them)
bvt_grid <- expand.grid(
  x = seq(-3, 3, length.out = 200), # seq builds a sequence vector starting from
  # 3 until 3 with step such that the number of elements in the vector is 200
  s = seq(-4, 4, length.out = 200)
)

# compute the density at each coordinate of the grid
probs <- dmvnorm(bvt_grid, m, vcov_mat)
bvt_grid %>%
  mutate(prob = probs) %>% # add a column (?dplyr::mutate)
  ggplot() +
  geom_contour(aes(x, s, z = prob)) + # or geom_contour_filled
  coord_fixed(xlim = c(-3, 3), ylim = c(-5, 5), ratio = -.7)
```



3.3.1 Exercises

Take two bivariate random variables (the same as before, X, S). Complete these tasks:

a. Compute $P(X < 0 \cap S < 0)$ with `pmvnorm`
b. Compute $P(X < 0 \cap S < 0)$ by simulation (Monte Carlo estimate)
c. Compute $P(X > 1 \cap S < 0)$ by simulation. Can you do it with `pmvnorm`? (hint: check `?pmvnorm`)

3.3.2 Solutions

a.

```
pmvnorm(upper = c(0, 0), mean = m, sigma = vcov_mat)
```

```
## [1] 0.3479566
## attr(,"error")
## [1] 1e-15
## attr(,"msg")
## [1] "Normal Completion"
```

b.

```
sim <- rmvnorm(100000, mean = m, sigma = vcov_mat) # simulate enough samples
# count all the observations satisfying the condition
# and divide by the number of total obs to obtain a ratio
```

```r
# mean() applied to logical values is the proportion of true vars
mean(sim[, 1] < 0 & sim[, 2] < 0)
```

```
## [1] 0.34856
```

c.

```r
mean(sim[, 1] > 1 & sim[, 2] < 0)
```

```
## [1] 0.0238
```

```r
# use lower and upper limits as described in the pmvnorm docs
pmvnorm(lower = c(1, -Inf), upper = c(Inf, 0), mean = m, sigma = vcov_mat)
```

```
## [1] 0.0240375
## attr(,"error")
## [1] 1e-15
## attr(,"msg")
## [1] "Normal Completion"
```

Increasing the Monte Carlo samples you will get a more accurate estimate of the probability measure.

Chapter 4

The linear model

During this lecture, we will use the *Advertising* dataset, which can be found in Kaggle. It shows, for each line, the revenue (`Sales`), which is the *response* depending on three *predictors*, i.e. money spent on three different advertising channels: `TV`, `Radio`, `Newspaper`. Every variable is continuous, allowing to explain the main R functions used in a plain linear regression task.

4.1 Dataset import

Import the dataset (you may have to select properly the current working directory). Remember a modern version of a dataset in R is called *tibble* (as opposed to the previous *dataframe*)

```
library(readr)
# here the wd contains a folder 'dataset' which
# in turn contains the file to be read
advertising <- read_csv("./datasets/advertising.csv")
```

```
## Rows: 200 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (4): TV, Radio, Newspaper, Sales
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
advertising
```

```
## # A tibble: 200 x 4
##       TV Radio Newspaper Sales
##    <dbl> <dbl>     <dbl> <dbl>
```

```
##  1 230.   37.8      69.2  22.1
##  2  44.5  39.3      45.1  10.4
##  3  17.2  45.9      69.3  12
##  4 152.   41.3      58.5  16.5
##  5 181.   10.8      58.4  17.9
##  6   8.7  48.9      75     7.2
##  7  57.5  32.8      23.5  11.8
##  8 120.   19.6      11.6  13.2
##  9   8.6   2.1       1     4.8
## 10 200.    2.6      21.2  15.6
## # i 190 more rows
```

4.2 Basic EDA (Exploratory Data Analysis)

Get some basic statistics with `summary()`

```
summary(advertising)
```

```
##        TV             Radio          Newspaper          Sales
##  Min.   :  0.70   Min.   : 0.000   Min.   :  0.30   Min.   : 1.60
##  1st Qu.: 74.38   1st Qu.: 9.975   1st Qu.: 12.75   1st Qu.:11.00
##  Median :149.75   Median :22.900   Median : 25.75   Median :16.00
##  Mean   :147.04   Mean   :23.264   Mean   : 30.55   Mean   :15.13
##  3rd Qu.:218.82   3rd Qu.:36.525   3rd Qu.: 45.10   3rd Qu.:19.05
##  Max.   :296.40   Max.   :49.600   Max.   :114.00   Max.   :27.00
```

You can `attach()` the dataset to access columns writing less code, although this is not recommended nowadays (see `with()` below). All the columns of the table are then available in the R environment without prepending the dataset name

```
attach(advertising)
head(TV)
```

```
## [1] 230.1  44.5  17.2 151.5 180.8   8.7
```

To revert back, detach the dataset

```
detach(advertising)
```

It is better to use `with()` instead. This function basically attaches a certain context object (e.g. the dataset namespace), executes the commands inside the curly brackets, and then detaches the context object.

```
with(advertising, {
  print(head(Sales))
  print(mean(TV))
  print(Radio[Radio < 20])
})
```

```
## [1] 22.1 10.4 12.0 16.5 17.9  7.2
## [1] 147.0425
##  [1] 10.8 19.6  2.1  2.6  5.8  7.6  5.1 15.9 16.9 12.6  3.5 16.7 16.0 17.4  1.5
## [16]  1.4  4.1  8.4  9.9 15.8 11.7  3.1  9.6 19.2  2.0 15.5  9.3 14.5 14.3  5.7
## [31]  1.6  7.7  4.1 18.4  4.9  1.5 14.0  3.5  4.3 10.1 17.2 11.0  0.3  0.4  8.2
## [46] 15.4 14.3  0.8 16.0  2.4 11.8  0.0 12.0  2.9 17.0  5.7 14.8  1.9  7.3 13.9
## [61]  8.4 11.6  1.3 18.4 18.1 18.1 14.7  3.4  5.2 10.6 11.6  7.1  3.4  7.8  2.3
## [76] 10.0  2.6  5.4  5.7  2.1 13.9 12.1 10.8  4.1  3.7  4.9  9.3  8.6
```

These are some of the functions that might be useful when gathering information about a dataset.

```r
length(advertising) # columns!
```

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

```r
nrow(advertising)
```

```
## [1] 200
```

```r
dim(advertising)
```

```
## [1] 200   4
```

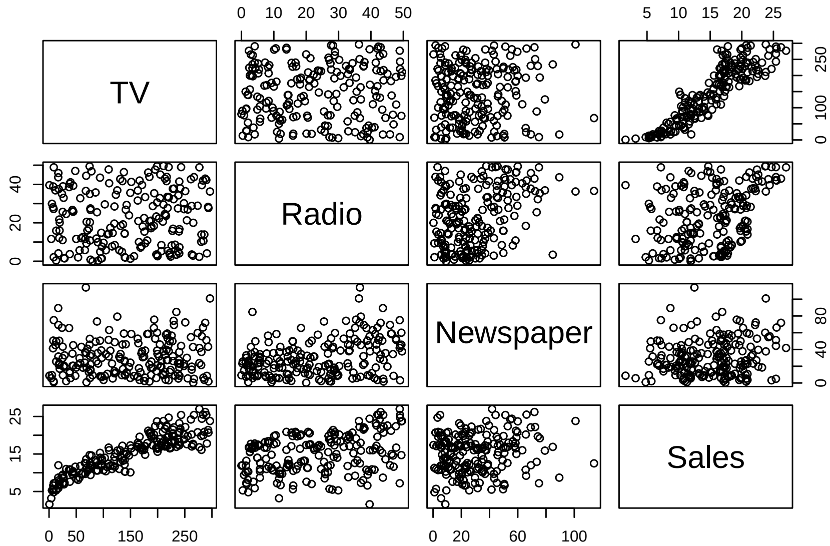
```r
names(advertising)
```

```
## [1] "TV"        "Radio"     "Newspaper" "Sales"
```

4.3 Simple plots

When exploring a dataset, `ggplot()` might be a bit excessive (recall that `ggplot()` is a modern way to draw cool plots using a graphical syntax). Faster plots can be drawn with `plot()` and `pairs()`

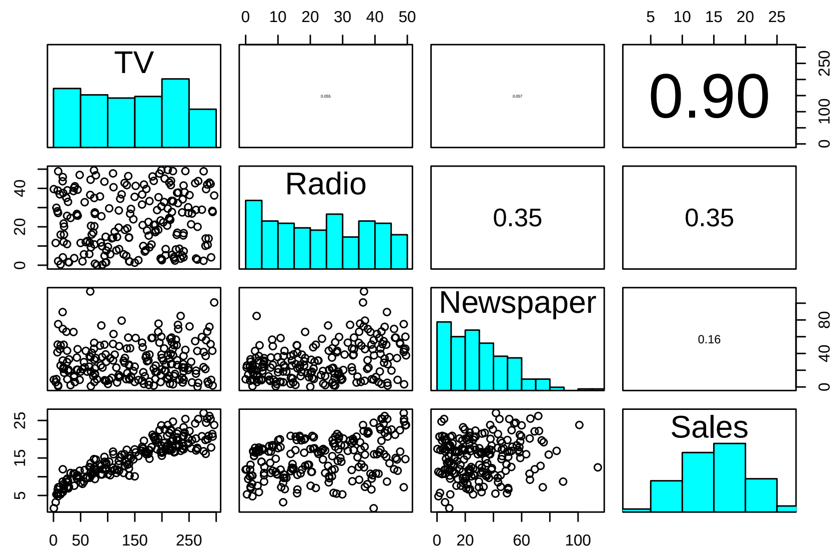
```r
plot(advertising)
```



A pair-plot gives a bigger picture of the cross correlations between variables. The base R `pairs()` function can be tweaked (check `?pairs`) in order to draw other kind of information of interest.

```
## put histograms on the diagonal
panel_hist <- function(x, ...) {
  usr <- par("usr")
  par(usr = c(usr[1:2], 0, 1.5))
  h <- hist(x, plot = FALSE)
  breaks <- h$breaks
  nb <- length(breaks)
  y <- h$counts
  y <- y / max(y)
  rect(breaks[-nb], 0, breaks[-1], y, col = "cyan", ...)
}
## put (absolute) correlations on the upper panels,
## with size proportional to the correlations.
panel_cor <- function(x, y, digits = 2, prefix = "", cex_cor, ...) {
  par(usr = c(0, 1, 0, 1))
  r <- abs(cor(x, y))
  txt <- format(c(r, 0.123456789), digits = digits)[1]
  txt <- paste0(prefix, txt)
  if (missing(cex_cor)) cex_cor <- 0.8 / strwidth(txt)
  text(0.5, 0.5, txt, cex = cex_cor * r)
}
```

```r
pairs(advertising,
  upper.panel = panel_cor, diag.panel = panel_hist
)
```

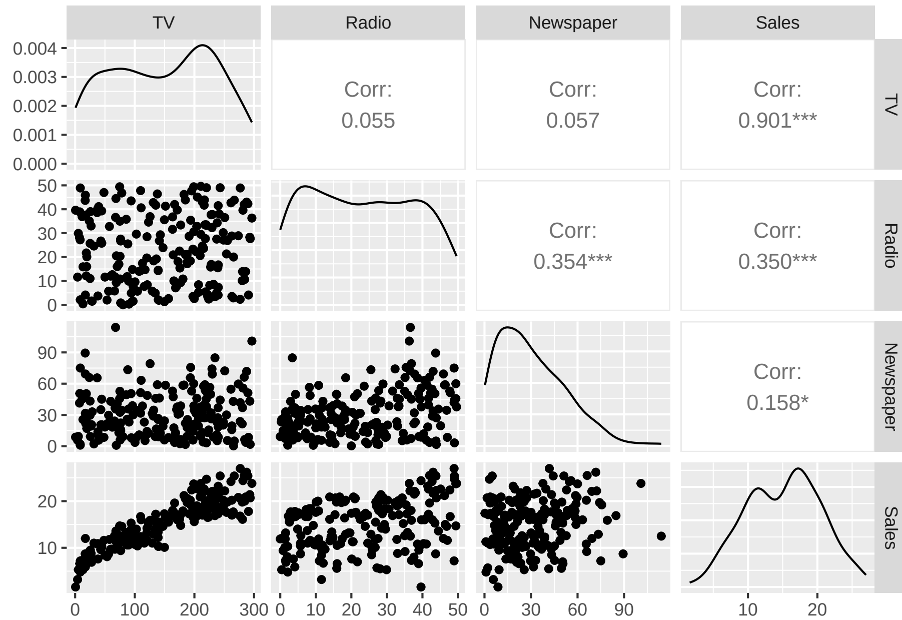


With much less effort, we can obtain a prettier version of the pair-plot.

```r
library(GGally)
```

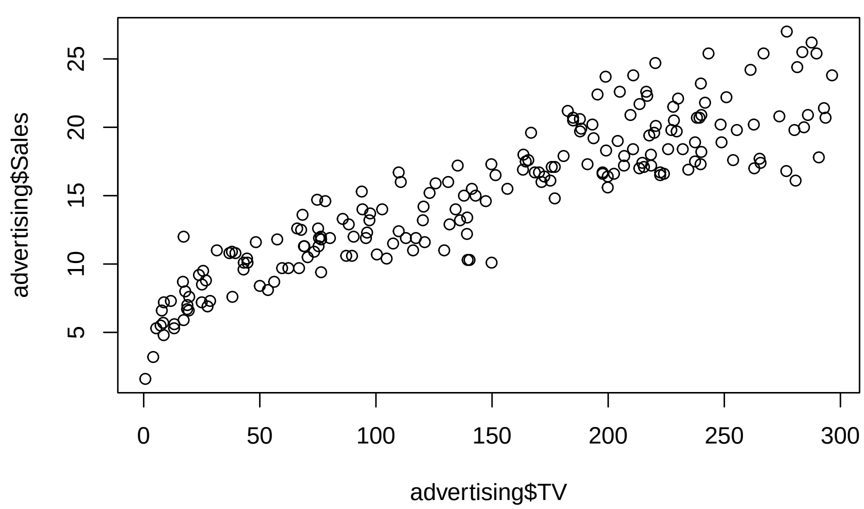
```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```r
ggpairs(advertising, progress = FALSE)
```



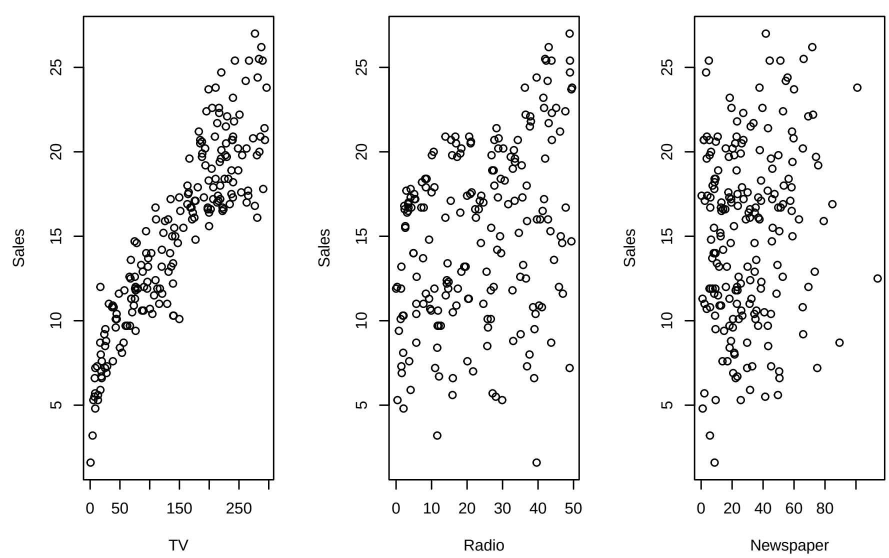
Again, with base R let's plot the predictors against the response. We know how to do that with a single predictor.

```
plot(advertising$TV, advertising$Sales)
```



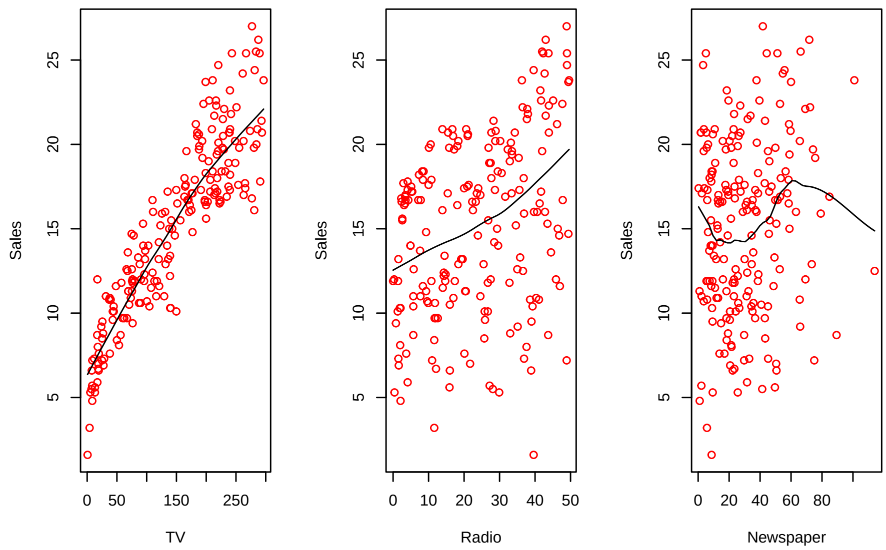
In case we want to add all plots to a single figure, we can do it as follow (base R)

```
# set plot parameters
with(advertising, {
  par(mfrow = c(1, 3))
  plot(TV, Sales)
  plot(Radio, Sales)
  plot(Newspaper, Sales)
})
```



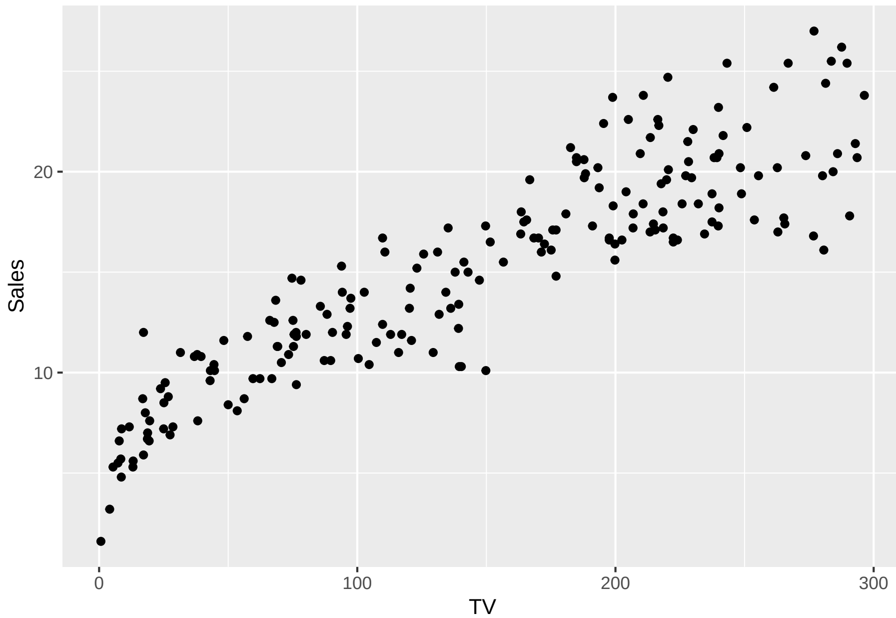
adding a smoothed line would look like this

```
# set plot parameters
with(advertising, {
  par(mfrow = c(1, 3))
  scatter.smooth(TV, Sales, col = "red")
  scatter.smooth(Radio, Sales, col = "red")
  scatter.smooth(Newspaper, Sales, col = "red", span = .3) # tweak with span
})
```



The same pictures can also be plotted with `ggplot()`, if preferred (in two ways).

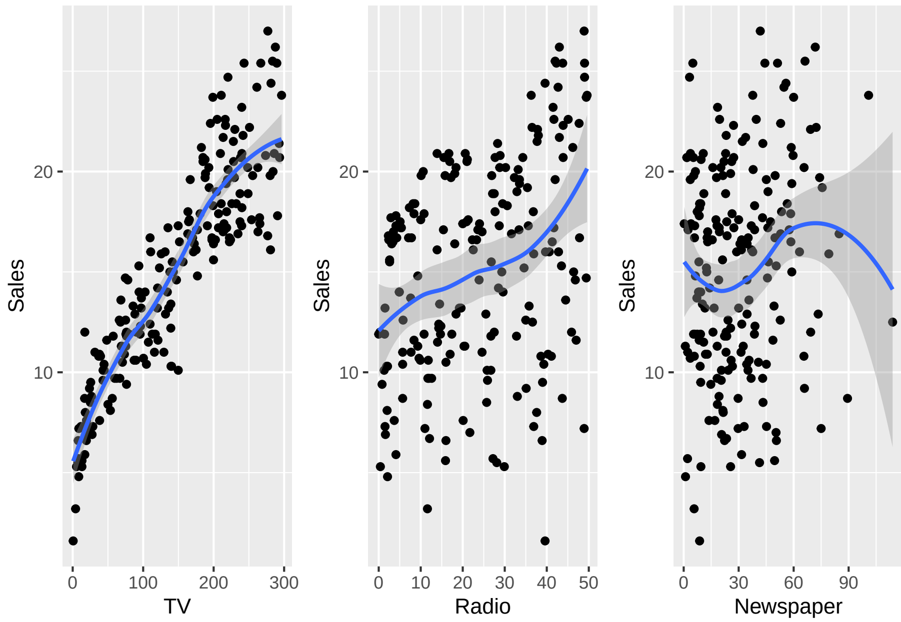
```
library(ggplot2)

ggplot(advertising) +
  geom_point(aes(TV, Sales))
```



First method: draw three separate plots and arrange them together with `ggarrange()`

```
library(ggpubr)
tvplt <- ggplot(advertising, mapping = aes(TV, Sales)) +
  geom_point() +
  geom_smooth()
radplt <- ggplot(advertising, mapping = aes(Radio, Sales)) +
  geom_point() +
  geom_smooth()
nwsplt <- ggplot(advertising, mapping = aes(Newspaper, Sales)) +
  geom_point() +
  geom_smooth()

ggarrange(tvplt, radplt, nwsplt,
  ncol = 3
)
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Second method: rearrange the dataset first, and feed everything to `ggplot()`.

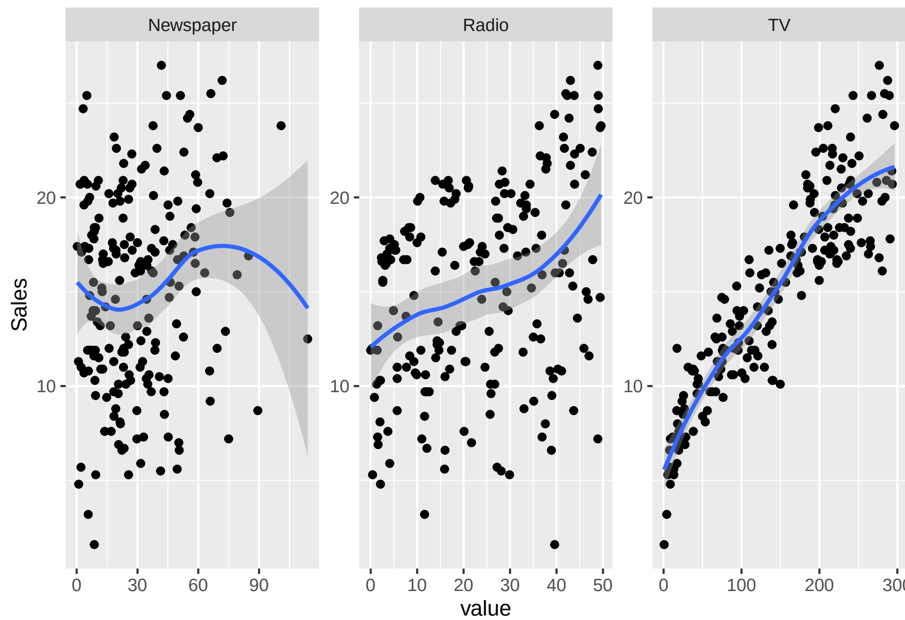
```
library(tibble)
library(tidyr)

## if using melt, need to switch to data.frame
# library(reshape2)
# advertising <- as.data.frame(advertising)
# long_adv <- melt(advertising, id.vars = "Sales")

long_adv <- advertising %>%
  gather(channel, value, -Sales)
head(long_adv)
```

```
## # A tibble: 6 x 3
##   Sales channel value
##   <dbl> <chr>   <dbl>
## 1  22.1 TV      230.
## 2  10.4 TV       44.5
## 3  12   TV       17.2
## 4  16.5 TV      152.
## 5  17.9 TV      181.
## 6   7.2 TV        8.7
```

```
ggplot(long_adv, aes(value, Sales)) +
  geom_point() +
```

```
geom_smooth() +
facet_wrap(~channel, scales = "free")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



If you're not sure why we need to use the `melt()` function, check the first lab (Iris dataset), or simply read the `?melt` helper.

4.4 Simple regression

The core of this class is the `lm()` function: its first argument is a *formula*, centered around the symbol ~, whith a response on its left and a list of predictors on its right.

Let's start with a simple single quantitative predictor linear model (simple linear regression)

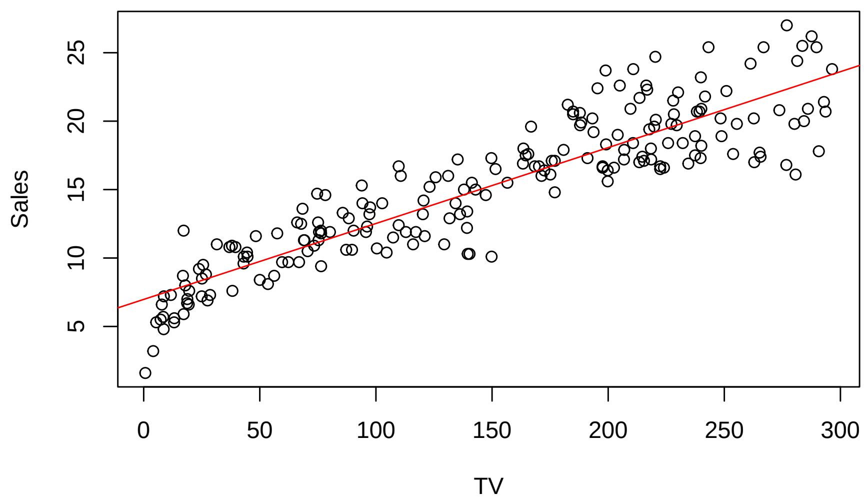
```
simple_reg <- lm(Sales ~ TV, data = advertising)
```

Now that we fitted the model, we have access to the coefficient estimates and we can plot the *estimated* regression line.

4.4.1 Plot

Here two ways of drawing the regression line, first in base R

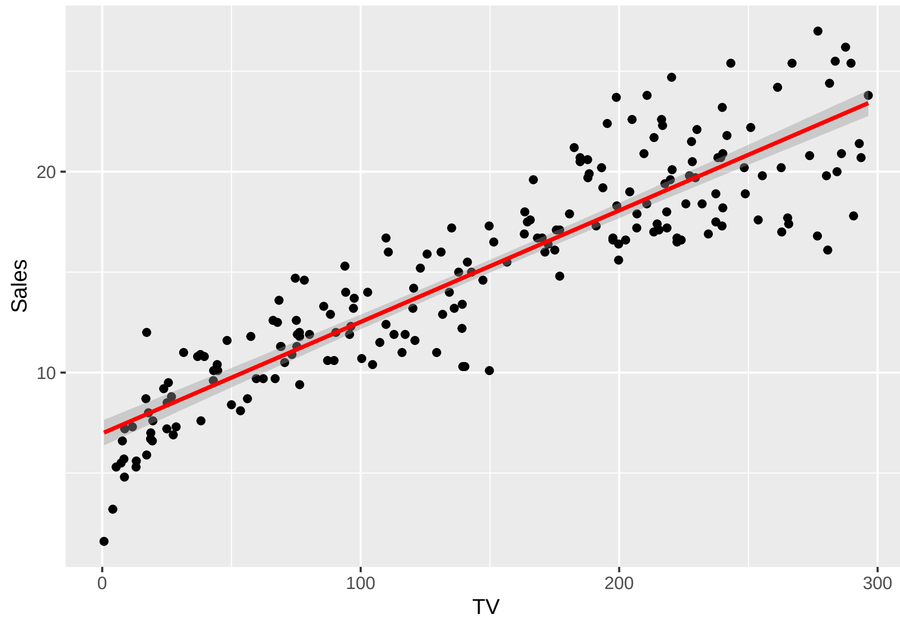
```r
with(data = advertising, {
  plot(x = TV, y = Sales)
  # abline draws a line from intercept and slope
  abline(
    simple_reg,
    col = "red"
  )
})
```



and in ggplot

```r
ggplot(simple_reg, mapping = aes(TV, Sales)) +
  geom_point() +
  geom_smooth(method = "lm", color = "red")
```

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



Actually, `lm()` does a lot more than just compute the least squares coefficients.

```r
summary(simple_reg)
```

```
##
## Call:
## lm(formula = Sales ~ TV, data = advertising)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.4438 -1.4857  0.0218  1.5042  5.6932
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 6.974821   0.322553   21.62   <2e-16 ***
## TV          0.055465   0.001896   29.26   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.296 on 198 degrees of freedom
## Multiple R-squared:  0.8122, Adjusted R-squared:  0.8112
## F-statistic: 856.2 on 1 and 198 DF,  p-value: < 2.2e-16
```

We will analyze in detail this output in the next subsection.

4.4.2 Summary

`summary(lm)` contains information about the following quantities:

- residuals: $Y - X\hat{\beta}$
- estimated coefficients: $\hat{\beta}$
- (estimated) standard errors on the coefficients (in the `Std. Error` column) $S\sqrt{(X'X)^{-1}_{i+1,i+1}}$ because $\frac{\hat{\beta}_i - \beta_i}{S\sqrt{(X'X)^{-1}}} \sim t(n-p)$
- in the `t value` column: value of the test statistic for the null hypothesis $H_0 : \beta_{i+1} = 0$
- in the `Pr(>|t|)` column: p-value regarding the null hypothesis above
- residual std error: $S = \sqrt{\frac{e'e}{n-p}}$, i.e. the square root of the Mean Square Residual (MSR)
- multiple R squared: *later*
- adj R squared: *later*

Let's compute them to make sure we understood the concepts

4.4.2.1 Residuals

Easy to retrieve them (actually, they are the realization of the residuals)

$$e = y - X\hat{\beta}$$

```
y <- advertising$Sales
x <- cbind(1, advertising$TV)
e <- y - x %*% simple_reg$coefficients
head(tibble(
  lm_res = simple_reg$residuals,
  manual_res = as.vector(e)
))
```

```
## # A tibble: 6 x 2
##   lm_res manual_res
##    <dbl>      <dbl>
## 1  2.36       2.36
## 2  0.957      0.957
## 3  4.07       4.07
## 4  1.12       1.12
## 5  0.897      0.897
## 6 -0.257     -0.257
```

4.4.2.2 Estimate

Estimates are just the $\hat{\beta}$ values for each predictor (plus intercept). They are computed with the closed form max likelihood formula.

$$\hat{\beta} = (X'X)^{-1}X'Y$$

```
beta_hat <- solve(t(x) %*% x) %*% t(x) %*% y
head(tibble(
  lm_coeff = simple_reg$coefficients,
  manual_coeff = as.vector(beta_hat)
))
```

```
## # A tibble: 2 x 2
##   lm_coeff manual_coeff
##      <dbl>        <dbl>
## 1   6.97         6.97
## 2   0.0555       0.0555
```

4.4.2.3 Standard Error

For each predictor, this quantifies the estimated variation in the beta estimator. The lower the standard error is, the higher is the accuracy of that particular coefficient. For predictor i, it is computed as

$$SE_i = S\sqrt{(X'X)^{-1}_{i+1,i+1}}$$

$i + 1$ is used here since first column if for the intercept β_0.

```
n <- nrow(x)
p <- ncol(x)
rms <- t(e) %*% e / (n - p)

# SE for TV
tv_se <- sqrt(rms * solve(t(x) %*% x)[2, 2])
tv_se
```

```
##             [,1]
## [1,] 0.001895551
```

4.4.2.4 T-value and p-value

These two are related to each other. The first is the test statistics value, under the null hypothesis $H_0 : \beta_i = 0$, for the variable

$$\frac{\hat{\beta}_i - \beta_i}{S\sqrt{(X'X)^{-1}}}$$

which is student-T distributed with $n-2$ degrees of freedom.

```
# for TV
t_val <- simple_reg$coefficients[2] / tv_se
t_val
```

```
##         [,1]
## [1,] 29.2605
```

And finally, the p-value is the probability on a $t(n-2)$ distribution of the statistic to be beyond the value actually observed. Remember that with *beyond* we mean on both sides of the distribution, since the alternative hypothesis $H_1 : \beta \neq 0$ is two-sided.

```
# multiply by two because the alternative hyp is two-sided
p_val <- 2 * pt(t_val, n - 2, lower.tail = FALSE)
p_val
```

```
##              [,1]
## [1,] 7.927912e-74
```

Here we cannot appreciate the manual computation since the p-value is very low (meaning that we can reject the null, favoring the alternative).

Exercise: You can try to compute this value manually on another simple regression model where we use `Radio` as predictor.

```
radio_simple_reg <- lm(Sales ~ Radio, data = advertising)
summary(radio_simple_reg)
```

```
##
## Call:
## lm(formula = Sales ~ Radio, data = advertising)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.5632  -3.5293   0.6714   4.2504   8.6796
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  12.2357     0.6535  18.724  < 2e-16 ***
## Radio         0.1244     0.0237   5.251 3.88e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.963 on 198 degrees of freedom
## Multiple R-squared:  0.1222, Adjusted R-squared:  0.1178
## F-statistic: 27.57 on 1 and 198 DF,  p-value: 3.883e-07
```

Chapter 5

Tools for linear regression analysis

We will still use the advertising dataset. Let's load it.

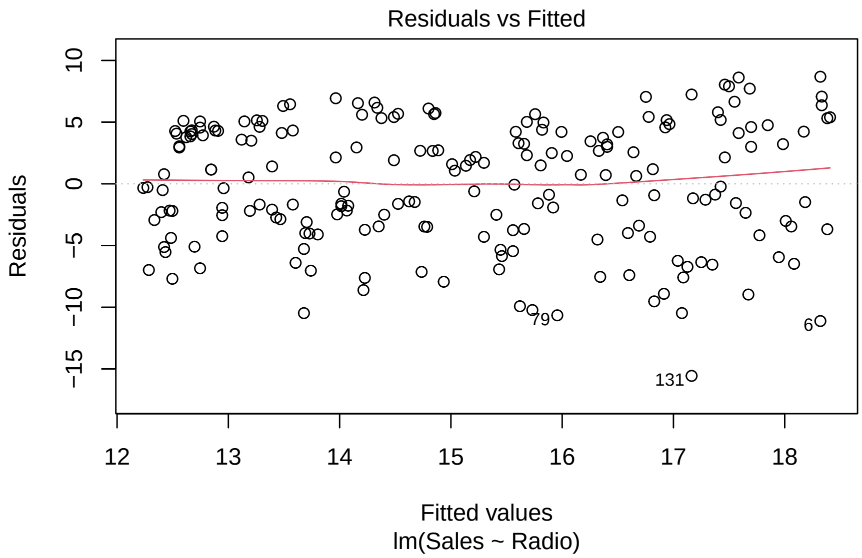
```
library(readr)
advertising <- read_csv("./datasets/advertising.csv")
```

```
## Rows: 200 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (4): TV, Radio, Newspaper, Sales
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

5.1 Plot

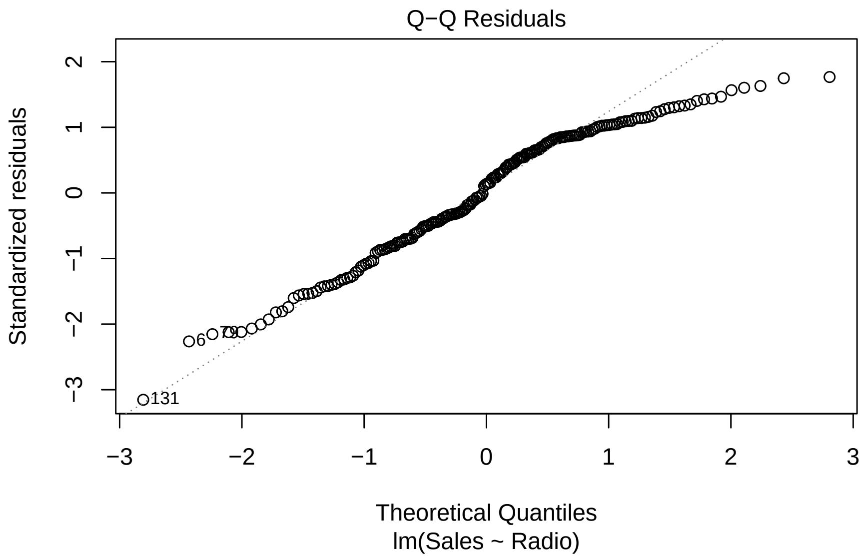
The plot function `plot()` provides special plots for a linear model object. See details `?plot.lm`.

```
radio_lm <- lm(Sales ~ Radio, data = advertising)
# residuals vs fitted
plot(radio_lm, which = 1)
```



This shows how the residuals e are distributed against the fitted values (or predictions) $\hat{y}$. Ideally, there should be no correlation, i.e. randomly spread around 0. If that is not the case and the plot shows a trend, the assumption of a linear regression model might not be appropriate.

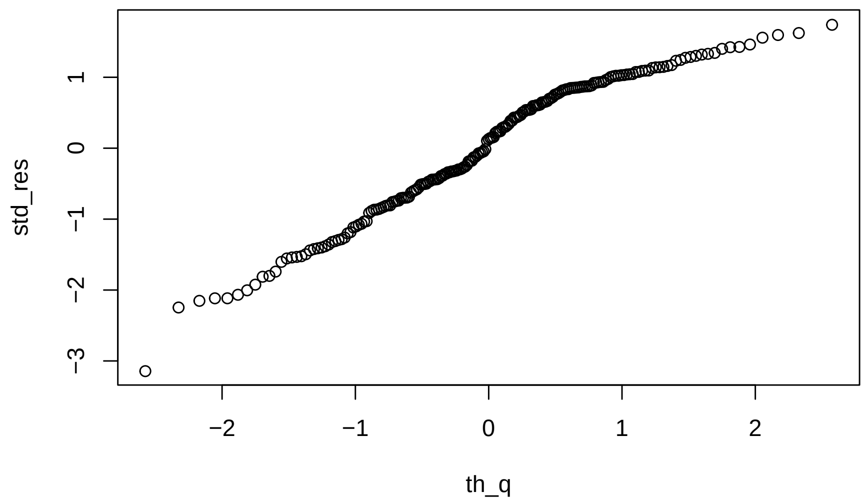
```
# normal q-q plot
plot(radio_lm, which = 2)
```



The Q-Q plot compares two distributions (or one theoretical distribution with an empirical one). Given some data, the question it helps answering to is: *is one distribution close to the other one?* In particular, the above plot is a Normal Q-Q plot of e, where its (standardized) distribution is compared to the standard normal distribution. More precisely, once the residuals are standardized (i.e. minus the mean, divided by the standard deviation) and sorted in ascending order, they are plotted against the theoretical quantiles of the standard Normal distributions. Example: the theoretical quantile of the k^{th} residual (in ascending order) is q_α where $\alpha = P(Z <= q_\alpha) = k/n$

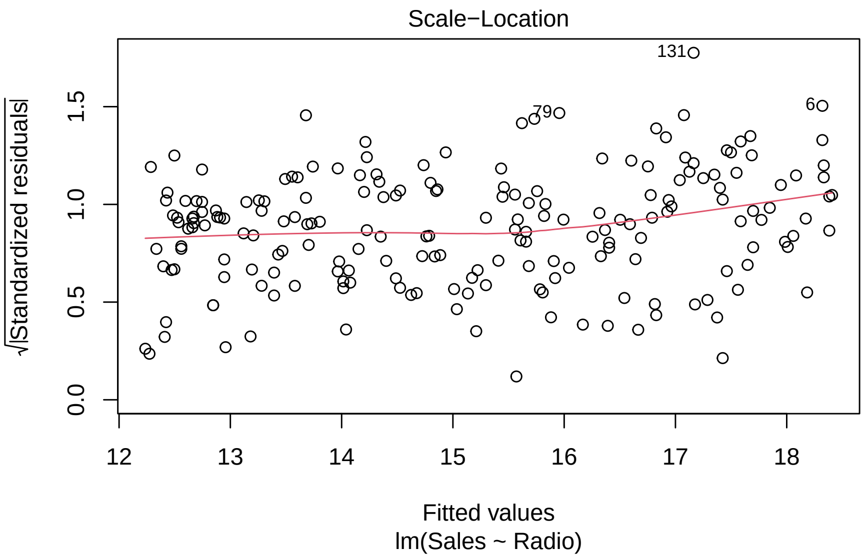
Exercise: draw a Q-Q plot "by hand":

```
n <- nrow(advertising)
std_res <- with(radio_lm, {
  sort((residuals - mean(residuals)) / sd(residuals),
    decreasing = FALSE
  )
})
th_q <- qnorm(1:n / n)
plot(th_q, std_res)
```



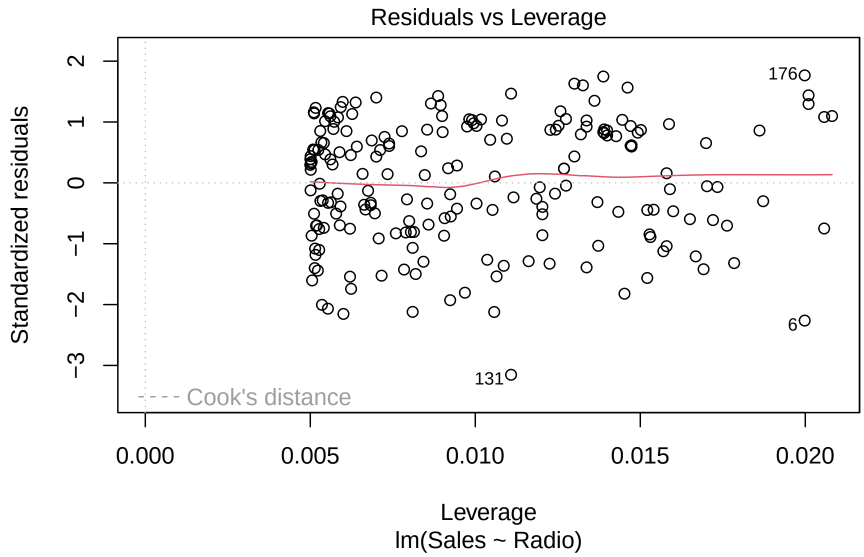
Note: the standardized residuals are actually computed slightly differently by the `plot.lm` R function, but this standardization is good enough for our purpose. Check the details in the help document `?plot.lm`.

```
# scale-location
plot(radio_lm, which = 3)
```



This plot only reduces the skewness of the standardized residuals by taking its square root.

```
# residuals vs leverage
plot(radio_lm, which = 5)
```



In order to understand this plot we have to introduced the so-called *hat matrix*

$$H = X(X'X)^{-1}X'$$

which is the projection matrix that maps Y to $\hat{Y}$. Its diagonal elements are particularly interesting and are called leverages of the observations. Note that $h_{ii} \in [0,1]$ and $\sum_i h_{ii} = p$, where p is the number of coefficients in the regression model. The leverage of an observation h_{ii} measures how much an observation y_i has an impact on the fitted value $\hat{y}_i$.[1]

5.2 Multiple predictors

Predictors in R linear models are defined in the formula.

What is a formula?

In R, `y ~ x + b` is a formula.

- `y` is the dependent variable(s) (e.g. response, label)
- `x + b` are the independent variables (e.g. predictors/features)
- can also be one-sided e.g. `~ x`
- `+` (plus) sign is not a sum, but a *join* operator
- when formulae are written, variables are not evaluated (symbolic model)
- you can exclude some terms explicitly by using `-` (e.g. `y ~ x - 1` will only estimate the `x` coefficient and not the intercept, which is implicitly added by `lm`)

More on formulae later (see Interactions in the next lesson).

Examples:

```r
f <- y ~ x + b
class(f)
```

```
## [1] "formula"
```

```r
f[[1]] # formula symbol
```

```
## `~`
```

```r
f[[2]] # dep vars
```

```
## y
```

```r
f[[3]] # indep vars
```

```
## x + b
```

[1] Explanation taken from https://it.mathworks.com/help/stats/hat-matrix-and-leverage.html

Generally, a linear model formula is composed by multiple predictors.

```r
complete_lm <- lm(Sales ~ TV + Radio + Newspaper, data = advertising)
# or
complete_lm <- lm(Sales ~ ., data = advertising)
```

The dot stands for all the possible predictors in the dataset.

```r
summary(complete_lm)
```

```
##
## Call:
## lm(formula = Sales ~ ., data = advertising)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.3034 -0.8244 -0.0008  0.8976  3.7473
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 4.6251241  0.3075012  15.041   <2e-16 ***
## TV          0.0544458  0.0013752  39.592   <2e-16 ***
## Radio       0.1070012  0.0084896  12.604   <2e-16 ***
## Newspaper   0.0003357  0.0057881   0.058    0.954
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.662 on 196 degrees of freedom
## Multiple R-squared:  0.9026, Adjusted R-squared:  0.9011
## F-statistic: 605.4 on 3 and 196 DF,  p-value: < 2.2e-16
```

5.3 Confidence regions

The hypothesis system setup (in the general case, multiple regression) is:

$$\begin{aligned} H_0 : &\quad C\beta = \theta \\ H_1 : &\quad C\beta \neq \theta \end{aligned}$$

The confidence region for β is given by the theorem by which:

$$\frac{(C\beta - C\hat{\beta})'(C(X'X)^{-1}C')^{-1}(C\beta - C\hat{\beta})}{q\text{MSR}} \sim F(q, n-p)$$

where $\text{rank}(C) = q \leq p$.

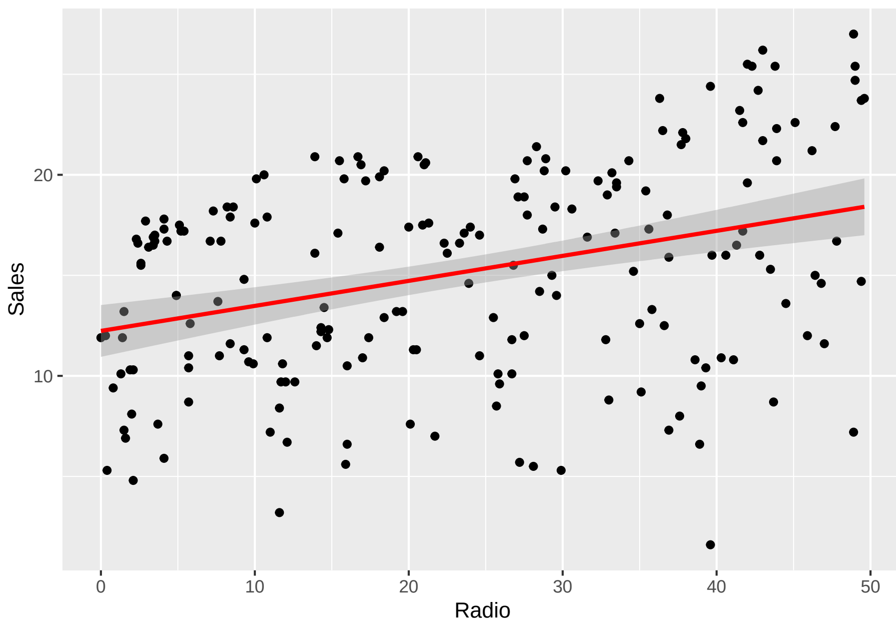
In case of $C = I$, the $(1-\alpha)$-confidence region is an ellipsoid in $\mathbb{R}^p$.

$$(\hat{\beta} - \beta)X'X(\hat{\beta} - \beta) \leq F_\alpha(p, n-p)p\text{MSR}$$

As already seen before, some plotting functions also draw a confidence region for the regression line.

```
library(ggplot2)
ggplot(radio_lm, mapping = aes(Radio, Sales)) +
  geom_point() +
  geom_smooth(method = "lm", color = "red")
```

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



This confidence region is drawn according to that formula.

To get the confidence intervals values, just use `confint()`

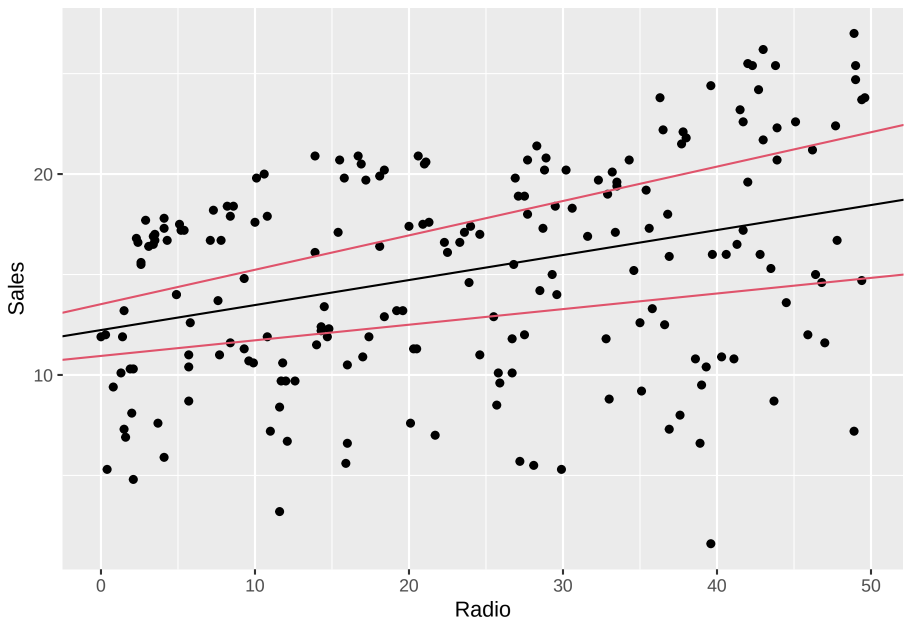
```
confint(radio_lm)
```

```
##                   2.5 %     97.5 %
## (Intercept) 10.94703557 13.5244084
## Radio        0.07770266  0.1711606
```

```
lb <- confint(radio_lm)[, 1]
ub <- confint(radio_lm)[, 2]
```

To plot it, e.g. using `ggplot`:

```r
# prevision region
ggplot(radio_lm) +
  geom_point(mapping = aes(Radio, Sales)) +
  geom_abline(
    intercept = radio_lm$coefficients[[1]],
    slope = radio_lm$coefficients[[2]], color = 1
  ) +
  geom_abline(intercept = lb[1], slope = lb[2], color = 2) +
  geom_abline(intercept = ub[1], slope = ub[2], color = 2)
```



The region between the lines is different (larger) from the one generated by `geom_smooth`. Without entering too much in detail, this happens because the confidence intervals in the second case are computed independently one coefficients from the other, while in ggplot's function the confidence region is jointly computed, therefore it's more accurate.

Chapter 6

Interactions and qualitative predictors

6.1 Transformations

Formulae in R linear models are much more powerful than what seen so far. In some cases we might be interested in transforming a variable before fitting the linear model.

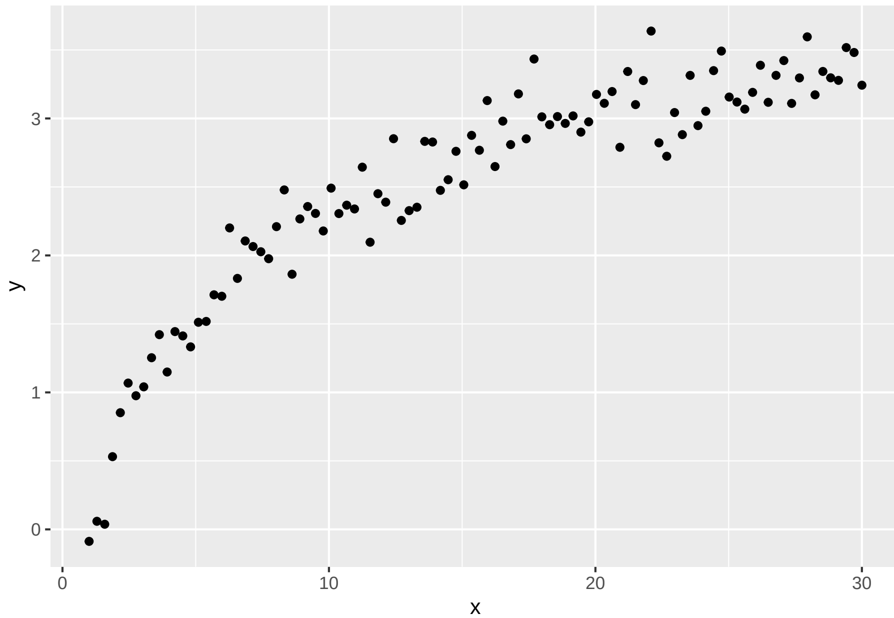
6.1.1 A simple example

For instance, let's take this synthetic dataset:

```r
library(tibble)
n <- 100
synth <- tibble(
  x = seq(from = 1, to = 30, length.out = n),
  y = log(x) + rnorm(n, 0, 0.2)
)
```

which looks like this:

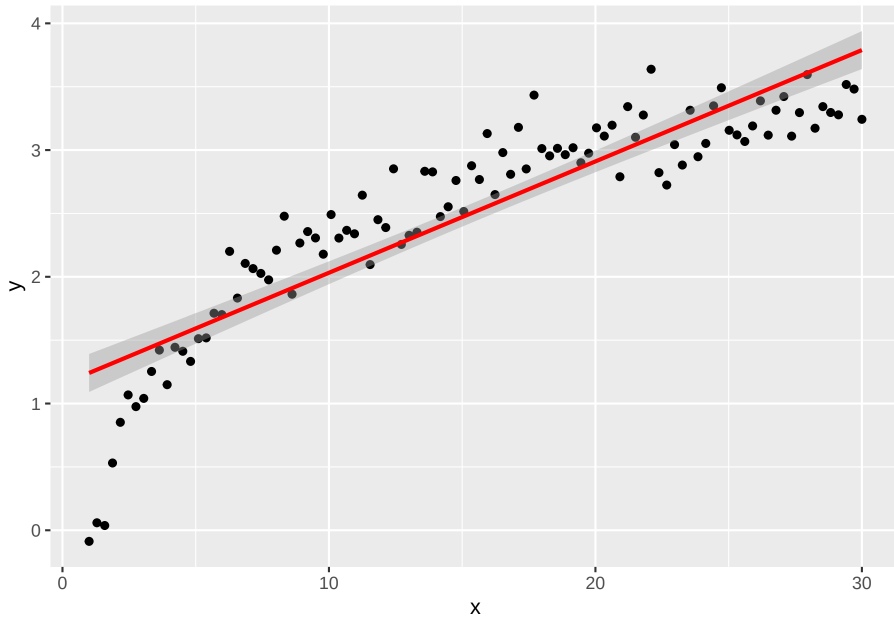
```r
library(ggplot2)
ggplot(synth) +
  geom_point(aes(x, y))
```



Of course, we can try to fit a linear model without any extra effort

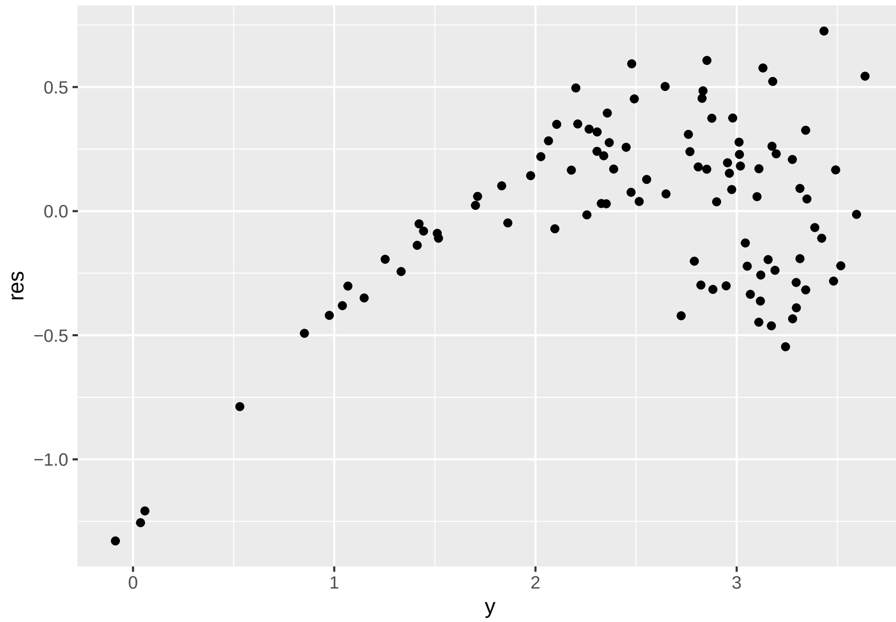
```
naive_lm <- lm(y ~ x, data = synth)
ggplot(naive_lm, mapping = aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", color = "red")
```

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



but if we plot the residuals, we can detect some issues for low values of y (definitely not uncorrelated).

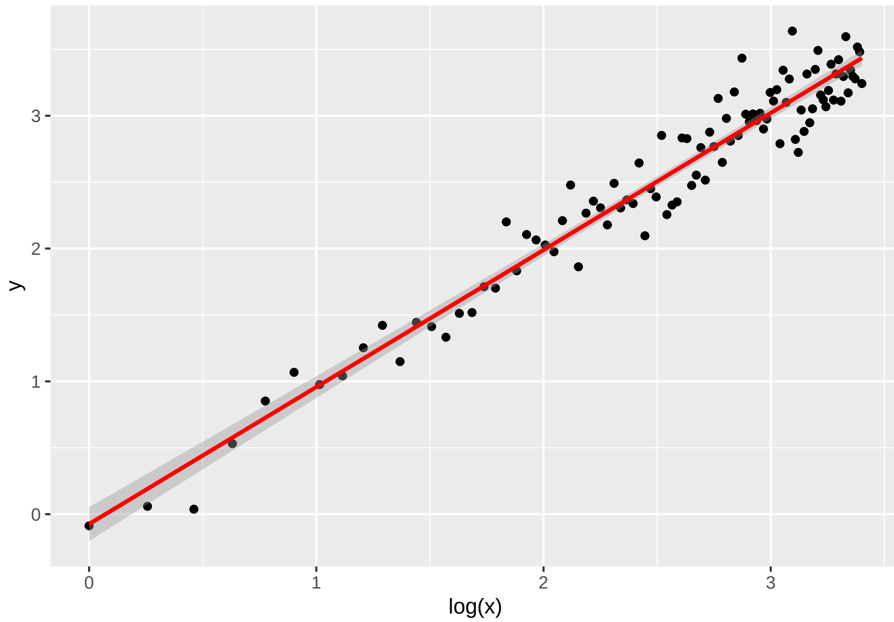
```
library(dplyr)
synth %>%
  mutate(res = naive_lm$residuals) %>% # adds the residual column
  ggplot() +
  geom_point(aes(y, res))
```



By understanding how x is distributed, we can fix this issue and fit the model on a transformation of itself, clearly $\log(x)$. We do this simply by adding the desired transformation in the formula, meaning that we don't have to tranform the dataset beforehand.

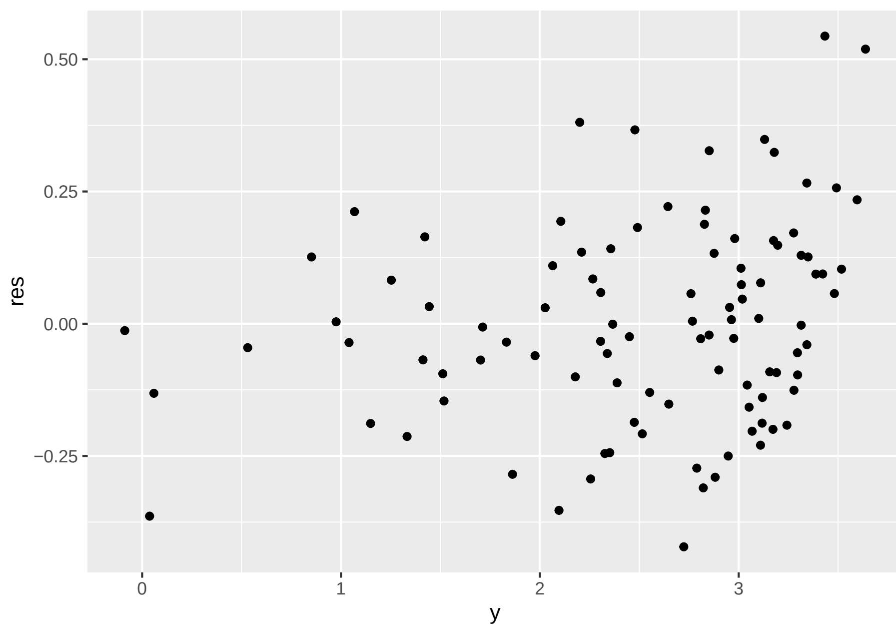
```
log_lm <- lm(y ~ log(x), data = synth)
ggplot(log_lm,
  mapping = aes(`log(x)`, y)
) + # notice the backticks!
  geom_point() +
  geom_smooth(method = "lm", color = "red")
```

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



and the residuals plot.

```
synth %>%
  mutate(res = log_lm$residuals) %>% # adds the residual column
  ggplot() +
  geom_point(aes(y, res))
```



6.1.2 On *advertising*

Back to our real dataset.

```
library(readr)
advertising <- read_csv("./datasets/advertising.csv")
```

```
## Rows: 200 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (4): TV, Radio, Newspaper, Sales
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

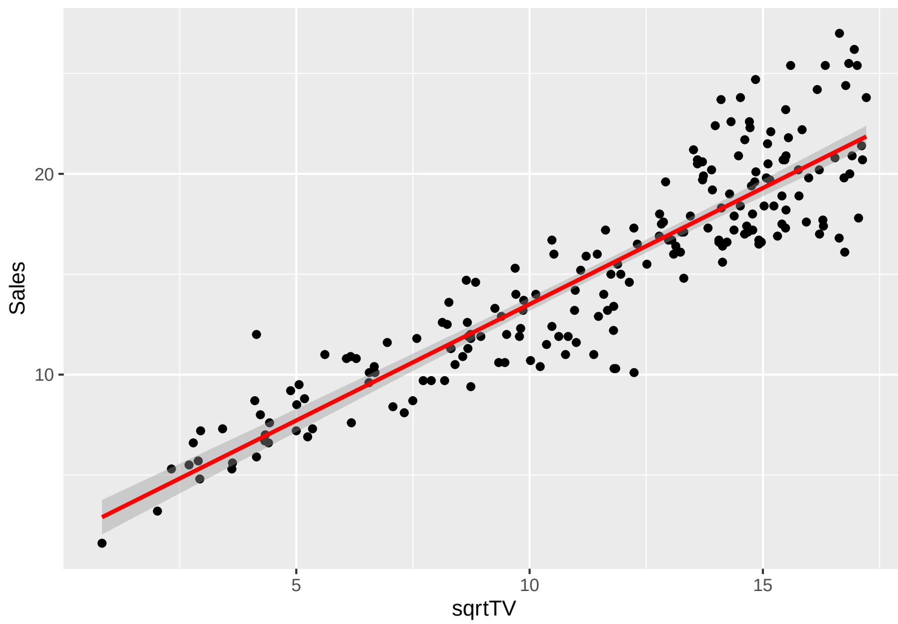
Let's try applying a transformation to the most promising predictor, in particular let's us $\sqrt{TV}$. The choice of the square root transformation comes from a first look at the scatter plot shown previously (TV against Sales), where we can detect a slightly curved trend which resembles a curve $y = \sqrt{x}$.

Let's plot the data after the transformation and observe that its trend better fit a straight line.

```
advertising %>%
  dplyr::select(Sales, TV) %>%
  dplyr::mutate(sqrtTV = sqrt(TV)) %>%
  ggplot(aes(sqrtTV, Sales)) +
```

```
  geom_point() +
  geom_smooth(method = "lm", color = "red")
```

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



We can fit a linear model. Its `summary` table will show a better R^2 score.

```
trans_lm <- lm(Sales ~ sqrt(TV), data = advertising)
```

> *Exercise*: plot the regression line and compare it with the simple model fitted on the raw data

The following two commands might help in inspecting a linear model with transformed data.

```
head(model.matrix(trans_lm)) # prints the design matrix
```

```
##   (Intercept)  sqrt(TV)
## 1           1 15.169047
## 2           1  6.670832
## 3           1  4.147288
## 4           1 12.308534
## 5           1 13.446189
## 6           1  2.949576
```

```
trans_lm$terms # inspect the linear model specifics
```

```
## Sales ~ sqrt(TV)
```

```
## attr(,"variables")
## list(Sales, sqrt(TV))
## attr(,"factors")
##          sqrt(TV)
## Sales           0
## sqrt(TV)        1
## attr(,"term.labels")
## [1] "sqrt(TV)"
## attr(,"order")
## [1] 1
## attr(,"intercept")
## [1] 1
## attr(,"response")
## [1] 1
## attr(,".Environment")
## <environment: R_GlobalEnv>
## attr(,"predvars")
## list(Sales, sqrt(TV))
## attr(,"dataClasses")
##     Sales  sqrt(TV)
## "numeric" "numeric"
```

6.2 Model selection

6.2.1 R-squared

One of the indicators computed with `summary` on a fitted linear model is the R^2 index (R-squared, or coefficient of determination). It is defined as

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

and shows the impact of the residuals proportionately to the variance of the response variable. It can be used to compare different models although it should not be considered as an absolute score of the model. The closer it is to 1, the better is the model fit.

This output, for instance, shows how the transformation used above seems to improve the accuracy of the regression task.

```
simple_lm <- lm(Sales ~ TV, data = advertising)
summary(simple_lm)$r.squared
```

```
## [1] 0.8121757
```

```
summary(trans_lm)$r.squared
```

```
## [1] 0.8216696
```

6.2.2 ANOVA

Another way of comparing two models, in particular one model with a smaller *nested* model, is the ANOVA test, which is an instance of the F-test, with statistics

$$F = \frac{\left(\frac{RSS_1 - RSS_2}{p_2 - p_1}\right)}{\left(\frac{RSS_2}{n - p_2}\right)}$$

where RSS is the residual sum of squares and $p_2 > p_1$ (model 1 is smaller). We know from theory that $F \sim F(p_2 - p_1, n - p_2)$

A low p-value for the F statistic means that we can reject the hypothesis that the smaller model explains the data well enough.

Here we compare the model with a single predictor (squared root of TV), which is the *smaller* model, with a model with also the Radio data as additional predictor.

```
double_lm <- lm(Sales ~ sqrt(TV) + Radio, data = advertising)
anova(trans_lm, double_lm)
```

```
## Analysis of Variance Table
##
## Model 1: Sales ~ sqrt(TV)
## Model 2: Sales ~ sqrt(TV) + Radio
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    198 990.80
## 2    197 410.88  1    579.92 278.04 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

> *Exercise*: compute both the F-statistics and the associated p-value "*by hand*" and verify your solution with the anova summary table.

6.3 Qualitative predictors

We recycle a textbook example taken from McClave JT., Benson PG. e Sincich T. (2014). *Statistics for Business and Economics.* Pearson Education Limited. We want to study the effect on the response (variable *distance*) of 4 different brands of golf ball (A,B,C,D) (variable *brand*) and the club type

(DRIVER/IRON) (variable *club*). These two features are qualitative predictors, also called factors. A robot player is used.

```
wide_golf <- read_tsv("./datasets/golfer.tsv")
```

```
## Rows: 8 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (1): club
## dbl (4): A, B, C, D
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
spec(wide_golf) # prints information about the detected data types
```

```
## cols(
##   club = col_character(),
##   A = col_double(),
##   B = col_double(),
##   C = col_double(),
##   D = col_double()
## )
```

Although it's not always necessary, it is always best to explicitly tell R to interpret qualitative predictors data as factors. This can be done with `read_tsv`, first by reading the data as it is and then calling the `spec()` function over the new tibble.

In this case, the output is saying that the first column has been detected as `numeric`, which is false, because the golfer number is just an identification number. Therefore we correct this by copying the output, manually editing the first column type from `col_double()` to `col_factor()` and setting the `read_tsv` parameter `col_types` to that.

```
wide_golf <- read_tsv("./datasets/golfer.tsv",
  col_types = cols(
    club = readr::col_factor(),
    A = col_double(),
    B = col_double(),
    C = col_double(),
    D = col_double()
  )
)

library(dplyr)
# need to switch from wide to long format
golf <- wide_golf %>%
```

```r
  gather(brand, distance, A:D)

## if using data.frame, you can use `melt()` from
## reshape2
# golf <- melt(wide_golf, id.vars = 1,
#              variable.name = "brand",
#              value.name = "distance")
```

Let us study first the effect of *brand* on the *distance*.

```r
lm_brand <- lm(distance ~ brand, data = golf)
summary(lm_brand)
```

```
##
## Call:
## lm(formula = distance ~ brand, data = golf)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -48.638 -31.703  -0.481  32.947  42.475
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  199.862     12.262  16.299 8.04e-16 ***
## brandB         8.338     17.341   0.481    0.634
## brandC         5.275     17.341   0.304    0.763
## brandD        -4.737     17.341  -0.273    0.787
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 34.68 on 28 degrees of freedom
## Multiple R-squared:  0.02322,    Adjusted R-squared:  -0.08143
## F-statistic: 0.2219 on 3 and 28 DF,  p-value: 0.8804
```

```r
model.matrix(lm_brand) # this is the design matrix
```

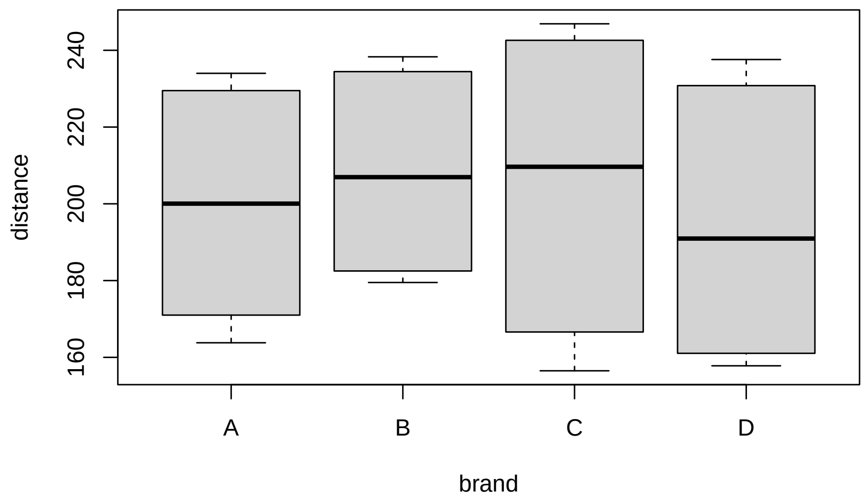
```
##    (Intercept) brandB brandC brandD
## 1            1      0      0      0
## 2            1      0      0      0
## 3            1      0      0      0
## 4            1      0      0      0
## 5            1      0      0      0
## 6            1      0      0      0
## 7            1      0      0      0
## 8            1      0      0      0
## 9            1      1      0      0
## 10           1      1      0      0
```

```
## 11          1       1       0       0
## 12          1       1       0       0
## 13          1       1       0       0
## 14          1       1       0       0
## 15          1       1       0       0
## 16          1       1       0       0
## 17          1       0       1       0
## 18          1       0       1       0
## 19          1       0       1       0
## 20          1       0       1       0
## 21          1       0       1       0
## 22          1       0       1       0
## 23          1       0       1       0
## 24          1       0       1       0
## 25          1       0       0       1
## 26          1       0       0       1
## 27          1       0       0       1
## 28          1       0       0       1
## 29          1       0       0       1
## 30          1       0       0       1
## 31          1       0       0       1
## 32          1       0       0       1
## attr(,"assign")
## [1] 0 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$brand
## [1] "contr.treatment"
```

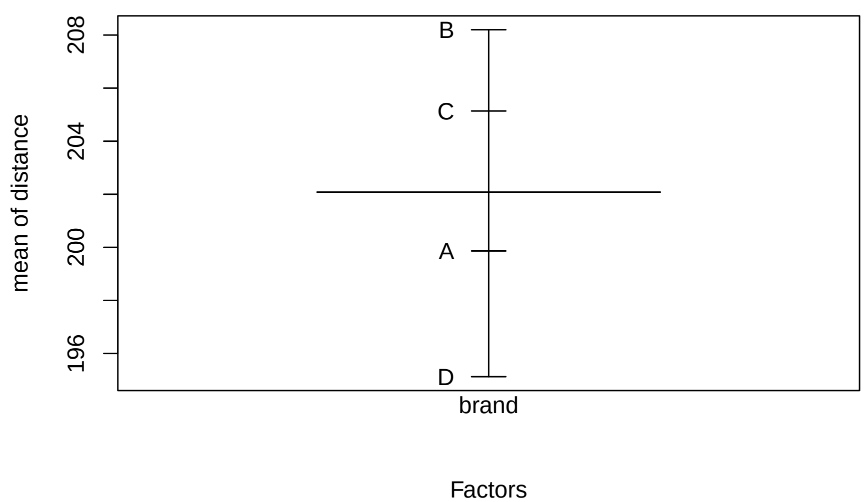
6.3.1 Data visualization

To visualize the data, we can use boxplots or more specialized graphics for qualitative predictors (for the latter, we have to define the predictors explicitly as factor objects in R). Recall that prettier graphics can always be produced using *ggplot2*.

```
boxplot(distance ~ brand, data = golf)
```




```
golf$brand <- as.factor(golf$brand)
plot.design(distance ~ brand, data = golf)
```



6.4 Interactions

Interactions are added in the `lm` formula. More specifically:

- `a:b` (colon op) includes the cross-variable between two predictors
- `a*b` (asterisk op) includes the two predictors individually and the cross-variable (i.e. writing `y ~ a + b + a:b` is equivalent to writing `y ~ a*b`)

Back to our dataset, let us now add the second factor *club* and build a 'small' additive model...

```
small_lm <- lm(distance ~ brand + club, data = golf)
summary(small_lm)
```

```
##
## Call:
## lm(formula = distance ~ brand + club, data = golf)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.9688  -5.2156   0.7375   5.2875  11.2063
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  231.531      3.032  76.371   <2e-16 ***
## brandB         8.338      3.835   2.174   0.0386 *
## brandC         5.275      3.835   1.376   0.1803
## brandD        -4.737      3.835  -1.235   0.2273
## clubIRON     -63.337      2.712 -23.358   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.67 on 27 degrees of freedom
## Multiple R-squared:  0.9539, Adjusted R-squared:  0.9471
## F-statistic: 139.8 on 4 and 27 DF,  p-value: < 2.2e-16
```

... and a larger one with interaction.

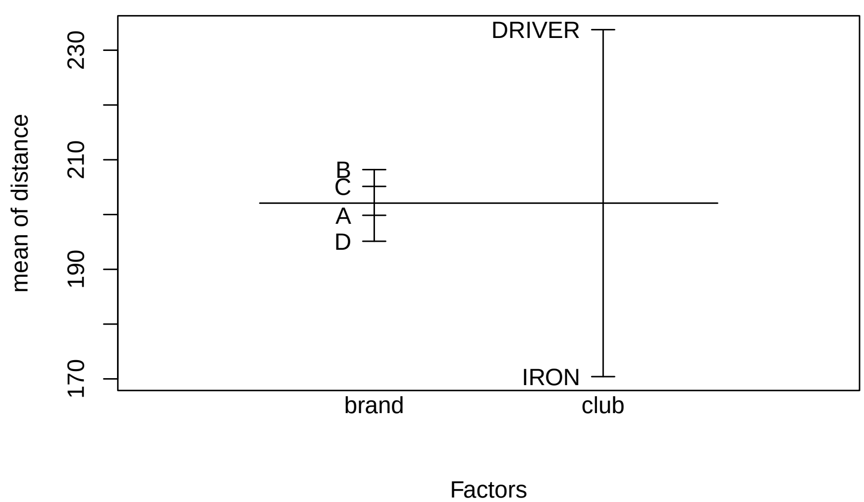
```
large_lm <- lm(distance ~ brand * club, data = golf)
summary(large_lm)
```

```
##
## Call:
## lm(formula = distance ~ brand * club, data = golf)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.6750  -2.7000   0.3125   3.4875  11.8250
##
```

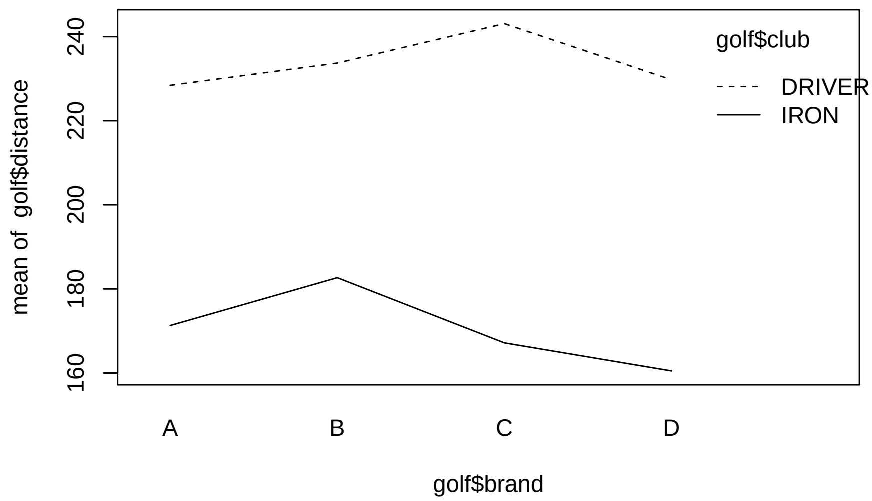
```
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      228.425      2.927  78.051  < 2e-16 ***
## brandB             5.300      4.139   1.281  0.21259
## brandC            14.675      4.139   3.546  0.00165 **
## brandD             1.325      4.139   0.320  0.75163
## clubIRON         -57.125      4.139 -13.802 6.55e-13 ***
## brandB:clubIRON    6.075      5.853   1.038  0.30966
## brandC:clubIRON  -18.800      5.853  -3.212  0.00373 **
## brandD:clubIRON  -12.125      5.853  -2.072  0.04923 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.853 on 24 degrees of freedom
## Multiple R-squared:  0.9762, Adjusted R-squared:  0.9692
## F-statistic: 140.4 on 7 and 24 DF,  p-value: < 2.2e-16
```

An interaction plot suggests interactions may be significant (the lines are not parallel), but we have to test for it since interaction plots are subject to sampling variation.

```r
plot.design(distance ~ brand * club, data = golf)
```




```r
interaction.plot(golf$brand, golf$club, golf$distance)
```



6.5 Model comparison

We can compare the complete model to a smaller one with ANOVA test.

```
anova(large_lm)
```

```
## Analysis of Variance Table
##
## Response: distance
##             Df Sum Sq Mean Sq  F value    Pr(>F)
## brand        3    801     267   7.7908 0.0008401 ***
## club         1  32093   32093 936.7516 < 2.2e-16 ***
## brand:club   3    766     255   7.4524 0.0010789 **
## Residuals   24    822      34
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

With the last *anova()* command we have built a more traditional anova table. We can compare the two models also with the *anova* command.

```
anova(small_lm, large_lm)
```

```
## Analysis of Variance Table
##
## Model 1: distance ~ brand + club
## Model 2: distance ~ brand * club
```

```
##   Res.Df     RSS Df Sum of Sq      F   Pr(>F)
## 1     27 1588.20
## 2     24  822.24  3    765.96 7.4524 0.001079 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Interactions are significant, meaning each combination of the two factors tells a different story.

```r
fitted(large_lm)
```

```
##       1       2       3       4       5       6       7       8       9      10
## 228.425 228.425 228.425 228.425 171.300 171.300 171.300 171.300 233.725 233.725
##      11      12      13      14      15      16      17      18      19      20
## 233.725 233.725 182.675 182.675 182.675 182.675 243.100 243.100 243.100 243.100
##      21      22      23      24      25      26      27      28      29      30
## 167.175 167.175 167.175 167.175 229.750 229.750 229.750 229.750 160.500 160.500
##      31      32
## 160.500 160.500
```

6.6 Predict

Especially when dealing with qualitative data, predictions for new unseen data can be easily computed with the `predict` function, which simply applies the regression coefficients to the provided data (arbitrarily generated below inside a tibble).

```r
predict.lm(small_lm,
  newdata = data.frame(
    club = c("DRIVER", "IRON", "IRON"),
    brand = c("A", "B", "B")
  ),
  interval = "confidence"
)
```

```
##        fit      lwr      upr
## 1 231.5312 225.3108 237.7517
## 2 176.5312 170.3108 182.7517
## 3 176.5312 170.3108 182.7517
```

```r
# for prediction intervals...
predict.lm(large_lm,
  newdata = data.frame(club = "DRIVER", brand = "A"),
  interval = "prediction",
  level = .99
)
```

```
##       fit      lwr      upr
```

```
## 1 228.425 210.1216 246.7284
```

Chapter 7

Generalized Linear Models (Binomial)

7.1 Students dataset

We want to analyze how students choose the study program from general, academic and technic (*vocation*)

- `ses`: socio-economic status
- `schtyp`: school type
- `read`, `write`, `math`, `science`: grade/score for each subject

```
library(readr)
# load the data
# tsv - similar format to csv
students <- read_delim("./datasets/students.tsv", delim = "\t", col_types = cols(
  id = col_double(),
  female = col_factor(),
  ses = col_factor(),
  schtyp = col_factor(),
  prog = col_factor(),
  read = col_double(),
  write = col_double(),
  math = col_double(),
  science = col_double()
))

# or with RData file
load("./datasets/students.RData")
head(students)
```

```
## # A tibble: 6 x 9
##      id female ses    schtyp prog      read write  math science
##   <dbl> <fct>  <fct>  <fct>  <fct>    <dbl> <dbl> <dbl>   <dbl>
## 1     1 female low    public vocation    34    44    40      39
## 2     2 female middle public vocation    39    41    33      42
## 3     3 male   low    public academic    63    65    48      63
## 4     4 female low    public academic    44    50    41      39
## 5     5 male   low    public academic    47    40    43      45
## 6     6 female low    public academic    47    41    46      40
```

7.2 EDA

As usual, a bit of data exploration before performing any statistical analysis.

```
# count the occurrences of all classes combinations (in prog and ses)
with(students, table(ses, prog))
```

```
##         prog
## ses      vocation academic general
##   low          12       19      16
##   middle       31       44      20
##   high          7       42       9
```

```
# using students dataframe attributes,
# call the rbind function using as arguments
# the result of the tapply operation,
# that is three vectors with mean and sd for the
# levels of `prog`
#
# the result is a 3x2 dataframe with mean and sd
# of the two columns
with(students, {
  do.call(rbind, tapply(
    write, prog,
    function(x) c(m = mean(x), s = sd(x))
  ))
})
```

```
##                 m        s
## vocation 46.76000 9.318754
## academic 56.25714 7.943343
## general  51.33333 9.397775
```

```
# or, with dplyr
library(dplyr)
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
students %>%
  group_by(prog) %>%
  summarise(mean = mean(write), sd = sd(write))
```

```
## # A tibble: 3 x 3
##   prog      mean    sd
##   <fct>    <dbl> <dbl>
## 1 vocation  46.8  9.32
## 2 academic  56.3  7.94
## 3 general   51.3  9.40
```

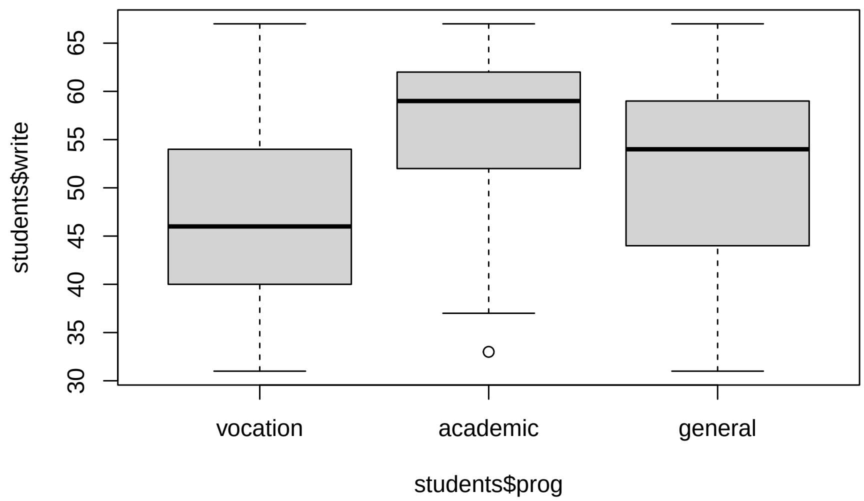
```
# simple way: do this for every program
mean(students$write[students$prog == "general"])
```

```
## [1] 51.33333
```

7.2.1 Plots

Boxplots allow to have a view of the distribution of a numeric variable over classes in a minimal representation. It shows first, second (median) and third quartiles, plus some outliers if present.

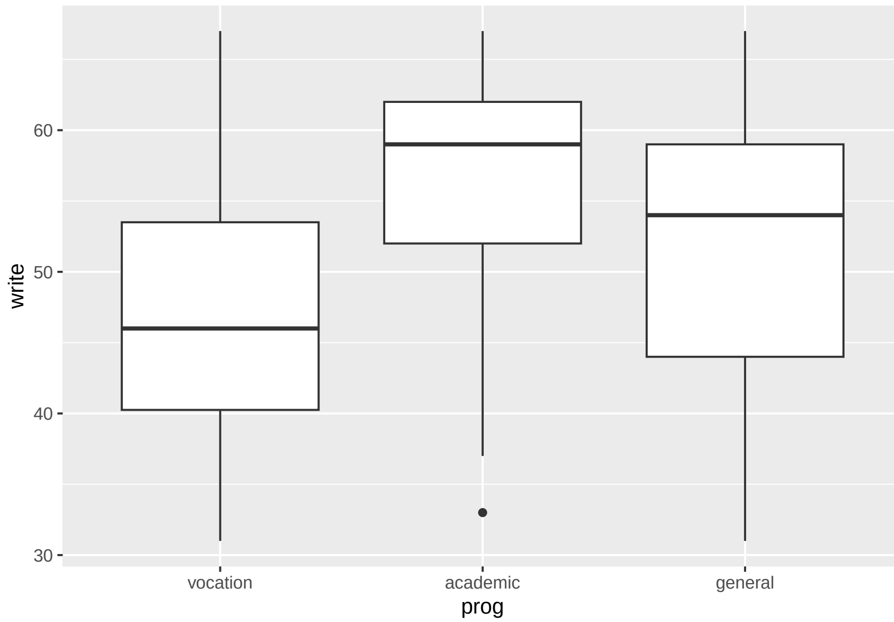
```
# boxplot in R
boxplot(students$write ~ students$prog)
```




```
# precise boundaries (numbers) are found with the `quantile()` function
with(students, {
  quantile(write[prog == "vocation"], prob = seq(0, 1, by = .25))
})
```

```
##    0%   25%   50%   75%  100%
## 31.00 40.25 46.00 53.50 67.00
```

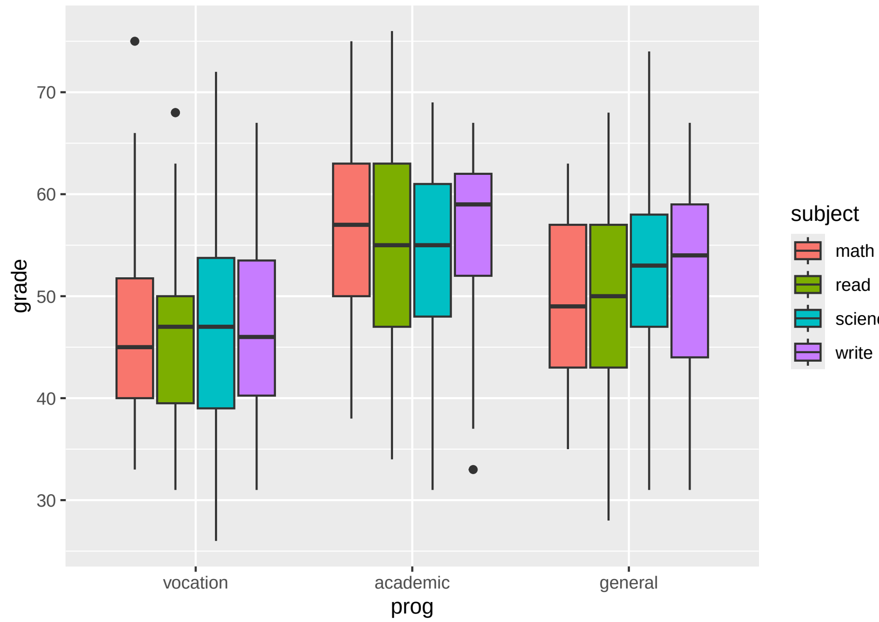
```
# boxplot in ggplot
library(ggplot2)
students %>%
  ggplot(aes(prog, write)) +
  geom_boxplot()
```




```
# geom_violin() # try also the "violin plot"
```

We can also put all subjects together, but we need to switch to long format with `gather` (or `reshape2::melt`).

```
# view of the grades distribution depending
# on subject and program
students %>%
  gather(key = "subject", value = "grade", write, science, math, read) %>%
  ggplot() +
  geom_boxplot(aes(prog, grade, fill = subject))
```




```
# or in different plots with
#    ...
#    geom_boxplot(aes(prog, grade)) +
#    facet_wrap(~ subject) # instead of
```

7.3 Test

We can further analyse the dataset attributes with some tests and traditional linear regression fit.

```
with(students %>% filter(prog != "vocation"), {
  tt_wp <- t.test(write[prog == "general"], # are the two prog distributed the same way?
    write[prog == "academic"],
    var.equal = TRUE
  )
  lm_wp <- summary(lm(write ~ prog)) # lm with qualitative predictor
  anova_wp <- summary(aov(write ~ prog)) # anova
  list(tt_wp, lm_wp, anova_wp)
})
```

```
## [[1]]
##
##  Two Sample t-test
##
## data:  write[prog == "general"] and write[prog == "academic"]
```

```
## t = -3.289, df = 148, p-value = 0.001256
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -7.882132 -1.965487
## sample estimates:
## mean of x mean of y
##  51.33333  56.25714
##
##
## [[2]]
##
## Call:
## lm(formula = write ~ prog)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -23.257  -4.257   2.705   5.743  15.667
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   56.257      0.820  68.610  < 2e-16 ***
## proggeneral   -4.924      1.497  -3.289  0.00126 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.402 on 148 degrees of freedom
## Multiple R-squared:  0.06811,    Adjusted R-squared:  0.06182
## F-statistic: 10.82 on 1 and 148 DF,  p-value: 0.001256
##
##
## [[3]]
##              Df Sum Sq Mean Sq F value  Pr(>F)
## prog          1    764   763.7   10.82 0.00126 **
## Residuals   148  10448    70.6
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

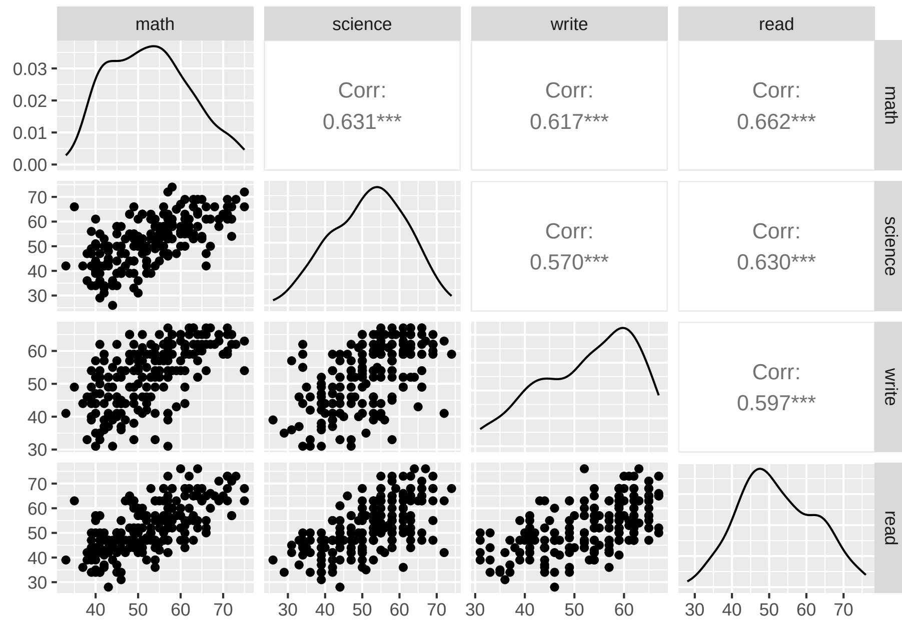
Notice how the T-test t-value is equal to the linear model coefficient estimate t-value. They are computed the same way.

7.4 Generalized Linear Model

The Xs have to be independent, thus we check the correlation plots.

```r
library(GGally)
students %>%
```

```
dplyr::select(math, science, write, read) %>%
ggpairs(progress = FALSE)
```



In order to be able to use the Binomial generalized linear model and set the program as response variable, we have to make a new dataset in which we define a binary class instead of a three levels factor. Here we arbitrarily choose to create a variable which is 1 for `vocation` and 0 for `general`.

```
# create a new dataframe
students_vg <- students %>%
  filter(prog != "academic") %>% # make distinction vocation-general only
  mutate(vocation = ifelse(prog == "vocation", 1, 0)) # transform class to binary

voc_glm <- glm(vocation ~ ses + schtyp + read + write + math, # choose some predictors
  data = students_vg, family = "binomial"
) # fit glm with binomial link
```

```
# new pipe operator (base R 4.2 or later) allows to send
# pipe results to any function parameter (not just the first one)
# and it's compatible with lm/glm calls (no need to create new datasets)
voc_glm <- students |>
  filter(prog != "academic") |>
  mutate(vocation = ifelse(prog == "vocation", 1, 0)) |>
  glm(vocation ~ ses + schtyp + read + write + math,
    data = _, family = "binomial"
```

```
  )
# `_` is placeholder for the piped dataframe
```

```
summary(voc_glm)
```

```
##
## Call:
## glm(formula = vocation ~ ses + schtyp + read + write + math,
##     family = "binomial", data = students_vg)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)    3.78961    1.62184   2.337   0.0195 *
## sesmiddle      1.04148    0.53206   1.957   0.0503 .
## seshigh        0.42122    0.68145   0.618   0.5365
## schtypprivate -1.06617    0.87344  -1.221   0.2222
## read          -0.02558    0.02940  -0.870   0.3843
## write         -0.02011    0.02836  -0.709   0.4782
## math          -0.04175    0.03438  -1.214   0.2246
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 131.43  on 94  degrees of freedom
## Residual deviance: 118.24  on 88  degrees of freedom
## AIC: 132.24
##
## Number of Fisher Scoring iterations: 4
```

In the summary output, few differences from the `lm` call can be noticed:

- the p-value for each coefficient is determined through a z-test instead of an exact t-test;
- R-squared cannot be computed (there are no residuals) and the *deviance* is printed instead:
  - `Null deviance` represents the distance of the null model (which has only the intercept) from a "perfect" saturated model
  - `Residual deviance` compares the fit with the saturated model (with number of parameters equal to the number of observations)

We can do the same thing with the pair `academic/general`.

```
students_ag <- students %>%
  filter(prog != "vocation") %>%
  mutate(academic = ifelse(prog == "academic", 1, 0))
```

```r
academic_glm <- glm(academic ~ ses + schtyp + read + write + math, # same predictors
  data = students_ag, family = "binomial"
)
summary(academic_glm)
```

```
##
## Call:
## glm(formula = academic ~ ses + schtyp + read + write + math,
##     family = "binomial", data = students_ag)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -5.07477    1.47499  -3.441 0.000581 ***
## sesmiddle      0.23316    0.47768   0.488 0.625471
## seshigh        0.77579    0.54950   1.412 0.158004
## schtypprivate  0.61998    0.53668   1.155 0.248007
## read           0.02441    0.02793   0.874 0.382069
## write          0.01130    0.02753   0.411 0.681416
## math           0.06720    0.03164   2.124 0.033689 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 183.26  on 149  degrees of freedom
## Residual deviance: 157.94  on 143  degrees of freedom
## AIC: 171.94
##
## Number of Fisher Scoring iterations: 4
```

Of course, we get different coefficient estimates with different models. We can compare them:

```r
cbind(
  summary(voc_glm)$coefficients[, c(1, 4)],
  summary(academic_glm)$coefficients[, c(1, 4)]
)
```

```
##                  Estimate   Pr(>|z|)    Estimate     Pr(>|z|)
## (Intercept)    3.78961265 0.01945937 -5.07476526 0.0005805405
## sesmiddle      1.04148131 0.05029279  0.23316035 0.6254706198
## seshigh        0.42121982 0.53649208  0.77579361 0.1580040841
## schtypprivate -1.06616814 0.22221864  0.61997866 0.2480066053
## read          -0.02557603 0.38428691  0.02441089 0.3820686083
## write         -0.02011257 0.47821104  0.01130034 0.6814162867
## math          -0.04175324 0.22462769  0.06720110 0.0336894120
```

Let's use `step` to chose the minimal set of useful predictors: it analyzes AIC for each combination of predictors, by progressively fitting a model with less and less predictors. The way it proceeds is the following:

1. fit the complete model,
2. for each of the predictors, fit another model with all but that predictor,
3. compare the AIC of all these models (`<none>` is the complete) and keep the one with the highest AIC;
4. repeat until the best model is found (i.e. `<none>` has highest AIC score)

Notice how this procedure can lead to sub-optimal models, since it doesn't try all possible predictors combinations, but rather finds a greedy solution to this search.

```
?step
```

```
step_voc <- step(voc_glm)
```

```
## Start:  AIC=132.24
## vocation ~ ses + schtyp + read + write + math
##
##          Df Deviance    AIC
## - write   1   118.74 130.74
## - read    1   119.00 131.00
## - math    1   119.75 131.75
## - schtyp  1   119.90 131.90
## <none>        118.24 132.24
## - ses     2   122.42 132.42
##
## Step:  AIC=130.74
## vocation ~ ses + schtyp + read + math
##
##          Df Deviance    AIC
## - read    1   120.25 130.25
## - schtyp  1   120.64 130.64
## <none>        118.74 130.74
## - math    1   121.07 131.07
## - ses     2   123.60 131.60
##
## Step:  AIC=130.25
## vocation ~ ses + schtyp + math
##
##          Df Deviance    AIC
## - schtyp  1   122.23 130.23
## <none>        120.25 130.25
## - ses     2   124.51 130.51
## - math    1   125.30 133.30
##
```

```
## Step:  AIC=130.23
## vocation ~ ses + math
##
##        Df Deviance    AIC
## <none>      122.23 130.23
## - ses   2   126.33 130.33
## - math  1   128.48 134.48
```

```
summary(step_voc)
```

```
##
## Call:
## glm(formula = vocation ~ ses + math, family = "binomial", data = students_vg)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.97676    1.41738   2.100   0.0357 *
## sesmiddle    0.97663    0.50784   1.923   0.0545 .
## seshigh      0.34949    0.66611   0.525   0.5998
## math        -0.07160    0.03016  -2.374   0.0176 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 131.43  on 94  degrees of freedom
## Residual deviance: 122.23  on 91  degrees of freedom
## AIC: 130.23
##
## Number of Fisher Scoring iterations: 4
```

```
# run this and check the results
step_academic <- step(academic_glm)
summary(step_academic)
```

7.4.1 Predictions

Working with generalized linear models, we can choose whether to get the logit estimate

$$g(\mu) = \eta = X\hat{\beta}$$

or the response probabilities, which is simply the inverse of the logit.

```
head(voc_glm$fitted.values)
```

```
##          1          2          3          4          5          6
```

```
## 0.5902369 0.8363280 0.6098686 0.6217618 0.7541615 0.7456350
```

```r
head(predict(voc_glm, newdata = students, type = "response")) # probs
```

```
##         1         2         3         4         5         6
## 0.5902369 0.8363280 0.2435802 0.4866921 0.4969118 0.4606502
```

```r
head(predict(voc_glm, newdata = students)) # logit
```

```
##           1           2           3           4           5           6
##  0.36494483  1.63115634 -1.13315009 -0.05324419 -0.01235304 -0.15772532
```

> Exercise: compute the inverse of the logit (manually) and verify that it equals the response found with `predict` (solution is in the Rmarkdown file).

The reason why we fitted two complementary models, is that we can combine the results to obtain predictions for both three programs together.

The logits are so defined for the two models:

$$X_{vg}\beta_{vg} = \log\left(\frac{\pi_v}{\pi_g}\right), X_{ag}\beta_{ag} = \log\left(\frac{\pi_a}{\pi_g}\right),$$

and knowing that $\pi_v + \pi_g + \pi_a = 1$ we have

$$\pi_g = \left(\frac{\pi_v}{\pi_g} + \frac{\pi_a}{\pi_g} + 1\right)^{-1}.$$

With some manipulation, replacing this result in the logits above, we can show that, for each class v, g, a:

$$\pi_v = \frac{e^{X_{vg}\beta_{vg}}}{1 + e^{X_{vg}\beta_{vg}} + e^{X_{ag}\beta_{ag}}}.$$

This formula is also called *softmax*, which converts numbers to probabilities (instead of just taking the max index, “hard”-max) and makes it possible to generalize from logistic regression to multiple category regression, sometimes called, indeed, *softmax regression*.

Let’s do this in R

```r
exp_voc <- exp(predict(voc_glm, type = "link", newdata = students))
exp_academic <- exp(predict(academic_glm, type = "link", newdata = students))
```

```r
norm_const <- 1 + exp_voc + exp_academic
pred <- tibble(
  pred_gen = 1, pred_voc = exp_voc,
  pred_acad = exp_academic
```

```
) / norm_const
head(pred)
```

```
##    pred_gen  pred_voc  pred_acad
## 1 0.3588022 0.5168311 0.12436676
## 2 0.1560455 0.7973583 0.04659614
## 3 0.3510001 0.1130281 0.53597187
## 4 0.4074066 0.3862819 0.20631153
## 5 0.3930219 0.3881968 0.21878130
## 6 0.3932633 0.3358800 0.27085675
```

Predictions must sum to 1 (they're normalized).

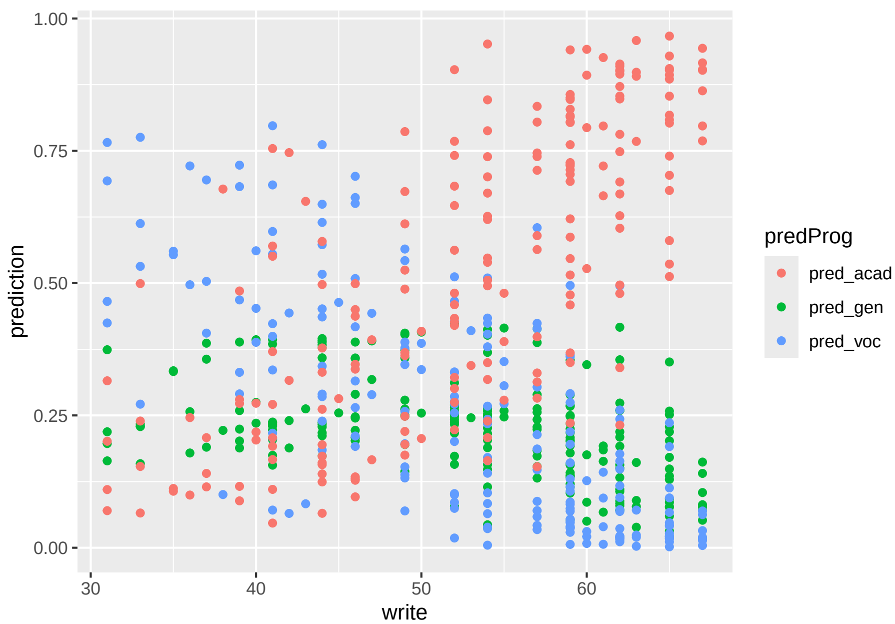
```
rowSums(pred)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

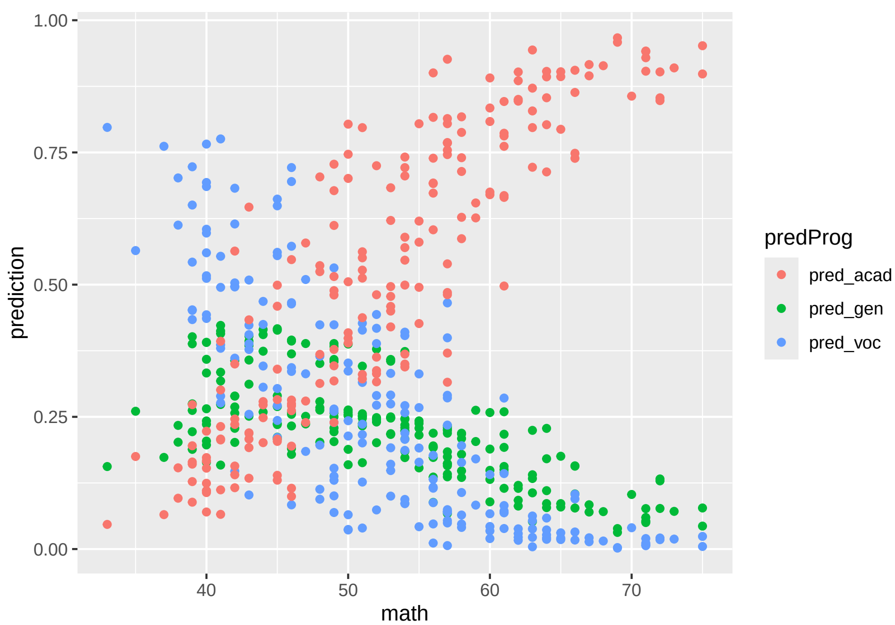
7.5 Graphic interpretation

These are some of the ways we can visualize the results. The plots interpretation is left as exercise.

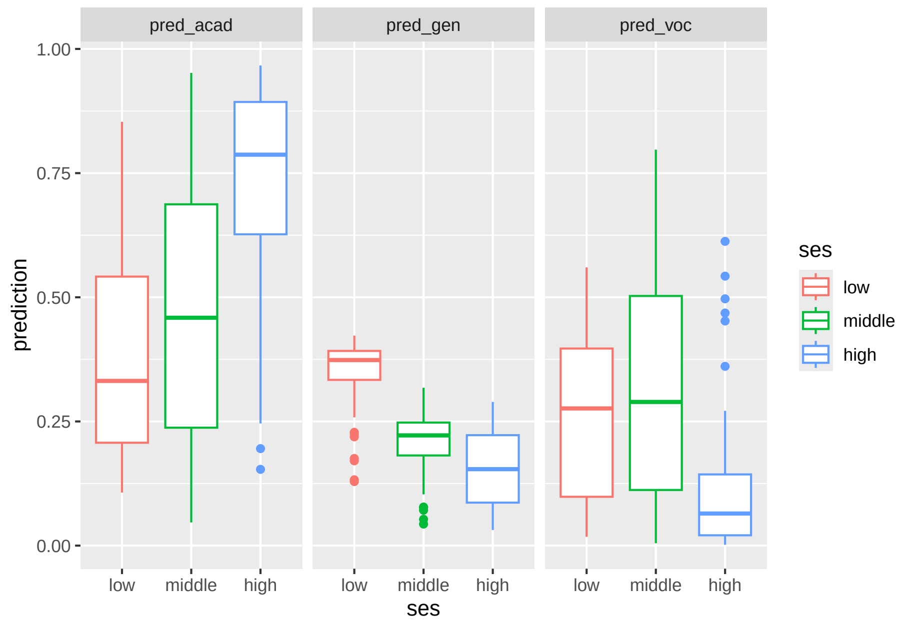
```
pred_stud_long <- bind_cols(pred, students) %>%
  gather(key = "predProg", value = "prediction", pred_gen:pred_acad)

pred_stud_long %>%
  ggplot() +
  geom_point(aes(write, prediction, color = predProg))
```




```
pred_stud_long %>%
  ggplot() +
  geom_point(aes(math, prediction, color = predProg))
```



```
pred_stud_long %>%
  ggplot() +
  geom_boxplot(aes(ses, prediction, color = ses)) +
  facet_wrap(~predProg)
```



7.6 Other tests

The models fitted so fare are not the only one that can give insights on the data. Here's some other models and tests made with arbitrary data. Feel free to further experiment the dataset.

```
# to run this, make sure you have R 4.2 installed.
# otherwise use the alternative way shown in the section above
general_glm <- students |>
  mutate(general = ifelse(prog == "general", 1, 0)) |>
  glm(general ~ ses + schtyp + read + write + math,
    data = _, family = "binomial"
  )
summary(general_glm)
```

```
##
## Call:
## glm(formula = general ~ ses + schtyp + read + write + math, family = "binomial",
##     data = mutate(students, general = ifelse(prog == "general",
##         1, 0)))
```

```
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)    0.888726   1.139382   0.780    0.435
## sesmiddle     -0.548233   0.411069  -1.334    0.182
## seshigh       -0.787325   0.503662  -1.563    0.118
## schtypprivate -0.050761   0.507839  -0.100    0.920
## read          -0.011416   0.024227  -0.471    0.637
## write          0.009743   0.024416   0.399    0.690
## math          -0.030597   0.027101  -1.129    0.259
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 213.27  on 199  degrees of freedom
## Residual deviance: 205.11  on 193  degrees of freedom
## AIC: 219.11
##
## Number of Fisher Scoring iterations: 4
```

```
general_alt_glm <- students |> # notice no filter on prog != "academic"
  mutate(general = ifelse(prog == "vocation", 1, 0)) |>
  glm(general ~ ses + read + write + math,
    data = _, family = "binomial"
  )
summary(general_alt_glm)
```

```
##
## Call:
## glm(formula = general ~ ses + read + write + math, family = "binomial",
##     data = mutate(students, general = ifelse(prog == "vocation",
##         1, 0)))
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  6.11922    1.38391   4.422 9.79e-06 ***
## sesmiddle    0.83768    0.45392   1.845   0.0650 .
## seshigh     -0.12410    0.58500  -0.212   0.8320
## read        -0.03261    0.02672  -1.220   0.2223
## write       -0.04151    0.02452  -1.693   0.0904 .
## math        -0.07799    0.03054  -2.554   0.0107 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 224.93  on 199  degrees of freedom
```

```
## Residual deviance: 179.08  on 194  degrees of freedom
## AIC: 191.08
##
## Number of Fisher Scoring iterations: 5
```

```
testdata <- tibble(
  ses = c("low", "middle", "high"),
  write = mean(students$write),
  math = mean(students$math),
  read = mean(students$read)
)
testdata %>%
  mutate(prob = predict(general_alt_glm, newdata = testdata, type = "response"))
```

```
## # A tibble: 3 x 5
##   ses    write  math  read  prob
##   <chr>  <dbl> <dbl> <dbl> <dbl>
## 1 low     52.8  52.6  52.2 0.132
## 2 middle  52.8  52.6  52.2 0.261
## 3 high    52.8  52.6  52.2 0.119
```

```
testdata <- tibble(
  ses = "low",
  write = c(30, 40, 50),
  math = mean(students$math),
  read = mean(students$read)
)
testdata %>%
  mutate(prob = predict(general_alt_glm, newdata = testdata, type = "response"))
```

```
## # A tibble: 3 x 5
##   ses   write  math  read  prob
##   <chr> <dbl> <dbl> <dbl> <dbl>
## 1 low      30  52.6  52.2 0.282
## 2 low      40  52.6  52.2 0.206
## 3 low      50  52.6  52.2 0.146
```

Chapter 8

Generalized Linear Models (Poisson)

8.1 Warpbreaks dataset

We want to analyse how the number of breaks in a wool thread depends on both the type of wool and the tension applied.

- `breaks`: number of breaks (integer)
- `wool`: wool type
- `tension`: tension level (low, medium or high)

This dataset is already embedded in base R, thus we don't need to read any external file and we can simply refer to it with its name.

```
head(warpbreaks)
```

```
##   breaks wool tension
## 1     26    A       L
## 2     30    A       L
## 3     54    A       L
## 4     25    A       L
## 5     70    A       L
## 6     52    A       L
```

```
summary(warpbreaks)
```

```
##      breaks      wool   tension
##  Min.   :10.00   A:27   L:18
##  1st Qu.:18.25   B:27   M:18
##  Median :26.00          H:18
##  Mean   :28.15
```

```
##   3rd Qu.:34.00
##   Max.   :70.00
```

8.2 EDA

This dataset is peculiar since we have two predictors, both qualitative. Some descriptive statistics might be useful.

Printing out the contingency table we observe that the dataset is balanced.

```
table(warpbreaks[, -1])
```

```
##     tension
## wool L M H
##    A 9 9 9
##    B 9 9 9
```

Moreover, we can inspect the response variable distribution along each combination of the two qualitative variables.

```
library(dplyr)
warpbreaks %>%
  group_by(wool, tension) %>%
  summarise(mean = mean(breaks), var = var(breaks))
```

```
## `summarise()` has grouped output by 'wool'. You can override using the
## `.groups` argument.
```

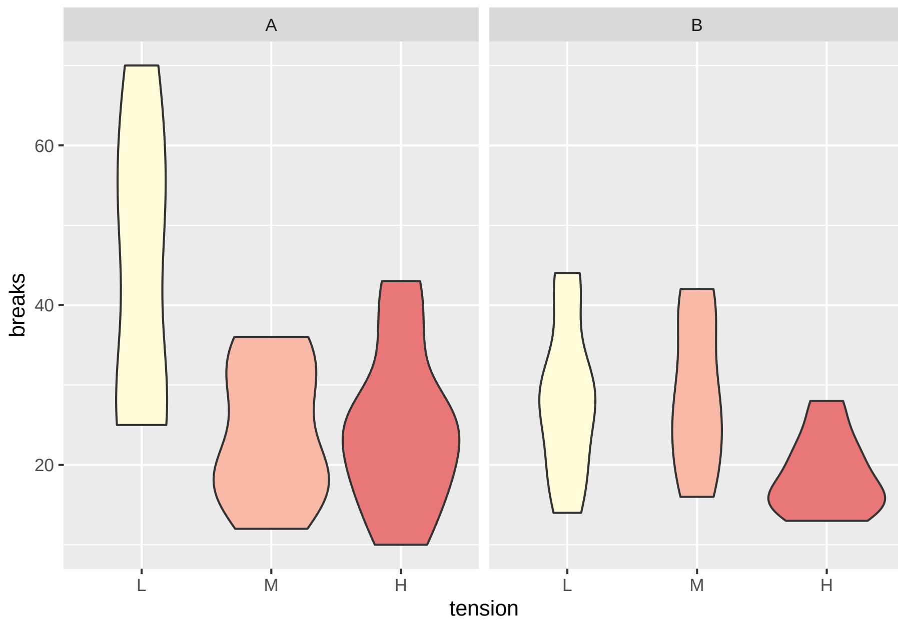
```
## # A tibble: 6 x 4
## # Groups:   wool [2]
##   wool  tension  mean   var
##   <fct> <fct>   <dbl> <dbl>
## 1 A     L        44.6 328.
## 2 A     M        24    75
## 3 A     H        24.6 106.
## 4 B     L        28.2  97.2
## 5 B     M        28.8  88.9
## 6 B     H        18.8  23.9
```

8.2.1 Plots

The above information can be easily visualized with boxplots. Below we see a variation of the boxplot, called *violin plot*, together with some fancy coloring which highlights the nature of the `tension` variable (discrete but with ordered levels, i.e. *low*, *medium*, *high*).

```
library(ggplot2)
warpbreaks %>%
  ggplot() +
```

```
  geom_violin(aes(tension, breaks, fill = as.integer(tension))) +
  scale_fill_gradient(name = "tension", low = "#FFFAD7", high = "#E97777") +
  facet_wrap(~wool) +
  theme(legend.position = "none")
```



8.3 Poisson Family GLM

Let's now fit a generalized linear model with log-link function (i.e. Poisson family).

```
breaks_glm <- glm(breaks ~ tension + wool,
  data = warpbreaks,
  family = "poisson"
)
summary(breaks_glm)
```

```
##
## Call:
## glm(formula = breaks ~ tension + wool, family = "poisson", data = warpbreaks)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.69196    0.04541  81.302  < 2e-16 ***
## tensionM    -0.32132    0.06027  -5.332 9.73e-08 ***
```

```
## tensionH    -0.51849    0.06396  -8.107 5.21e-16 ***
## woolB       -0.20599    0.05157  -3.994 6.49e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 297.37  on 53  degrees of freedom
## Residual deviance: 210.39  on 50  degrees of freedom
## AIC: 493.06
##
## Number of Fisher Scoring iterations: 4
```

The output is similar to the one from the Binomial family seen in the previous lecture, although here all the coefficients are related to discrete variables.

8.3.1 Response

The link function is simply

$$\eta_i = g(\mu_i) = \log(\mu_i)$$

therefore, it's enough to exponentiate the logits to obtain the response.

```r
breaks_glm$fitted.values
```

```
##        1        2        3        4        5        6        7        8
## 40.12354 40.12354 40.12354 40.12354 40.12354 40.12354 40.12354 40.12354
##        9       10       11       12       13       14       15       16
## 40.12354 29.09722 29.09722 29.09722 29.09722 29.09722 29.09722 29.09722
##       17       18       19       20       21       22       23       24
## 29.09722 29.09722 23.89035 23.89035 23.89035 23.89035 23.89035 23.89035
##       25       26       27       28       29       30       31       32
## 23.89035 23.89035 23.89035 32.65424 32.65424 32.65424 32.65424 32.65424
##       33       34       35       36       37       38       39       40
## 32.65424 32.65424 32.65424 32.65424 23.68056 23.68056 23.68056 23.68056
##       41       42       43       44       45       46       47       48
## 23.68056 23.68056 23.68056 23.68056 23.68056 19.44298 19.44298 19.44298
##       49       50       51       52       53       54
## 19.44298 19.44298 19.44298 19.44298 19.44298 19.44298
```

```r
# equals to
exp(predict(breaks_glm, newdata = warpbreaks))
```

```
##        1        2        3        4        5        6        7        8
## 40.12354 40.12354 40.12354 40.12354 40.12354 40.12354 40.12354 40.12354
##        9       10       11       12       13       14       15       16
```

```
## 40.12354 29.09722 29.09722 29.09722 29.09722 29.09722 29.09722 29.09722
##       17       18       19       20       21       22       23       24
## 29.09722 29.09722 23.89035 23.89035 23.89035 23.89035 23.89035 23.89035
##       25       26       27       28       29       30       31       32
## 23.89035 23.89035 23.89035 32.65424 32.65424 32.65424 32.65424 32.65424
##       33       34       35       36       37       38       39       40
## 32.65424 32.65424 32.65424 32.65424 23.68056 23.68056 23.68056 23.68056
##       41       42       43       44       45       46       47       48
## 23.68056 23.68056 23.68056 23.68056 23.68056 19.44298 19.44298 19.44298
##       49       50       51       52       53       54
## 19.44298 19.44298 19.44298 19.44298 19.44298 19.44298
```

8.3.2 Contrasts

We can use contrasts to compare the coefficients and understand whether two levels of a single variable report significantly different effects on the response variable. For instance, below we compare the *medium* tension level with the *high* tension level.

The reason why we use contrasts is that it provides statistically relevant information on the difference between two coefficients. I.e. looking at the GLM summary, we observe that the difference between `M` and `H` is 0.1971681, but how do we know if it's statistically relevant?

```
library(contrast)
cont <- contrast(breaks_glm,
  list(tension = "M", wool = "A"),
  list(tension = "H", wool = "A"),
  type = "individual"
)
# X = TRUE prints the design matrix used
print(cont, X = TRUE)
```

```
## glm model parameter contrast
##
##   Contrast       S.E.      Lower     Upper    t df Pr(>|t|)
##  0.1971681 0.06833267 0.05991786 0.3344183 2.89 50   0.0058
##
## Contrast coefficients:
##  (Intercept) tensionM tensionH woolB
##            0        1       -1     0
```

A low p-value lets us reject the hypothesis that the two coefficients are equal.

> Note: the `X = TRUE` parameter is actually a parameter of the `contrast` object, which tells the print function to show the contrast coefficients used.

Remember that the data is transformed into a model matrix by the `glm` function and it can be retrieved as follows.

```
# extract the dummy variables dataset
x <- model.matrix(breaks_glm)
head(x)
```

```
##   (Intercept) tensionM tensionH woolB
## 1           1        0        0     0
## 2           1        0        0     0
## 3           1        0        0     0
## 4           1        0        0     0
## 5           1        0        0     0
## 6           1        0        0     0
```

Let's compute the contrast output values manually:

```
# constants interc, tensM, tensH, woolB
coeff <- breaks_glm$coefficients
v <- c(0, 1, -1, 0) # contrast coefficients
coeff %*% v # contrast (difference between coeffs)
```

```
##           [,1]
## [1,] 0.1971681
```

```
# covariance matrix of the coefficients
covmat_j <- solve(t(x) %*% diag(breaks_glm$weights) %*% x)
```

Weights are related to each combination of the (qualitative) predictors. E.g. every row with the same predictor values will have the same weight.

```
dif <- v %*% coeff
se <- sqrt(v %*% covmat_j %*% v)
se # standard error
```

```
##            [,1]
## [1,] 0.06833267
```

```
tvalue <- dif / se
tvalue # t-statistic
```

```
##          [,1]
## [1,] 2.885414
```

```
df <- nrow(x) - ncol(x)
df # degrees of freedom of the T-student variable
```

```
## [1] 50
```

The p-value is computed taking the two extremes (bilateral).

```
pt(tvalue, df, lower.tail = FALSE) + pt(-tvalue, df, lower.tail = TRUE)
```

```
##             [,1]
## [1,] 0.005755553
```

We can provide multiple levels at once to contrasts:

```
cont_multi <- contrast(breaks_glm,
  list(tension = c("H", "M"), wool = "A"),
  list(tension = c("M", "L"), wool = "A"),
  type = "individual"
)
print(cont_multi, X = TRUE)
```

```
## glm model parameter contrast
##
##    Contrast       S.E.      Lower       Upper     t df Pr(>|t|)
##  -0.1971681 0.06833267 -0.3344183 -0.05991786 -2.89 50   0.0058
##  -0.3213204 0.06026580 -0.4423679 -0.20027301 -5.33 50   0.0000
##
## Contrast coefficients:
##  (Intercept) tensionM tensionH woolB
##            0       -1        1     0
##            0        1        0     0
```

8.4 Tests

Since we know that the difference of the deviances is Chi-squared distributed, we can perform some tests.

```
str(summary(breaks_glm)) # to view the attribute names
```

```
## List of 17
##  $ call          : language glm(formula = breaks ~ tension + wool, family = "poisson", da
##  $ terms         :Classes 'terms', 'formula'  language breaks ~ tension + wool
##   .. ..- attr(*, "variables")= language list(breaks, tension, wool)
##   .. ..- attr(*, "factors")= int [1:3, 1:2] 0 1 0 0 0 1
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:3] "breaks" "tension" "wool"
##   .. .. .. ..$ : chr [1:2] "tension" "wool"
##   .. ..- attr(*, "term.labels")= chr [1:2] "tension" "wool"
##   .. ..- attr(*, "order")= int [1:2] 1 1
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(breaks, tension, wool)
##   .. ..- attr(*, "dataClasses")= Named chr [1:3] "numeric" "factor" "factor"
```

```
##   .. .. ..- attr(*, "names")= chr [1:3] "breaks" "tension" "wool"
##  $ family        :List of 13
##   ..$ family    : chr "poisson"
##   ..$ link      : chr "log"
##   ..$ linkfun   :function (mu)
##   ..$ linkinv   :function (eta)
##   ..$ variance  :function (mu)
##   ..$ dev.resids:function (y, mu, wt)
##   ..$ aic       :function (y, n, mu, wt, dev)
##   ..$ mu.eta    :function (eta)
##   ..$ initialize:  expression({  if (any(y < 0))  stop("negative values not allowed for t
##   ..$ validmu   :function (mu)
##   ..$ valideta  :function (eta)
##   ..$ simulate  :function (object, nsim)
##   ..$ dispersion: num 1
##   ..- attr(*, "class")= chr "family"
##  $ deviance      : num 210
##  $ aic           : num 493
##  $ contrasts     :List of 2
##   ..$ tension: chr "contr.treatment"
##   ..$ wool   : chr "contr.treatment"
##  $ df.residual   : int 50
##  $ null.deviance : num 297
##  $ df.null       : int 53
##  $ iter          : int 4
##  $ deviance.resid: Named num [1:54] -2.38 -1.67 2.08 -2.57 4.26 ...
##   ..- attr(*, "names")= chr [1:54] "1" "2" "3" "4" ...
##  $ coefficients  : num [1:4, 1:4] 3.692 -0.3213 -0.5185 -0.206 0.0454 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:4] "(Intercept)" "tensionM" "tensionH" "woolB"
##   .. ..$ : chr [1:4] "Estimate" "Std. Error" "z value" "Pr(>|z|)"
##  $ aliased       : Named logi [1:4] FALSE FALSE FALSE FALSE
##   ..- attr(*, "names")= chr [1:4] "(Intercept)" "tensionM" "tensionH" "woolB"
##  $ dispersion    : num 1
##  $ df            : int [1:3] 4 50 4
##  $ cov.unscaled  : num [1:4, 1:4] 0.00206 -0.00153 -0.00153 -0.00119 -0.00153 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:4] "(Intercept)" "tensionM" "tensionH" "woolB"
##   .. ..$ : chr [1:4] "(Intercept)" "tensionM" "tensionH" "woolB"
##  $ cov.scaled    : num [1:4, 1:4] 0.00206 -0.00153 -0.00153 -0.00119 -0.00153 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:4] "(Intercept)" "tensionM" "tensionH" "woolB"
##   .. ..$ : chr [1:4] "(Intercept)" "tensionM" "tensionH" "woolB"
##  - attr(*, "class")= chr "summary.glm"
```

```
d0 <- summary(breaks_glm)$null.deviance # Msat - Mnull
df0 <- summary(breaks_glm)$df.null
d1 <- summary(breaks_glm)$deviance # Msat - Mfit
df1 <- summary(breaks_glm)$df.residual
```

```
deltaD <- d0 - d1 # chisq statistic of the model (Mfit - Mnull)
dfD <- df0 - df1 # chisq degrees of freedom
pchisq(deltaD, dfD, lower.tail = FALSE)
```

```
## [1] 9.750414e-19
```

Since the p-value is very low, we can reject the hypothesis that the fitted model is equal to a null model, thus the model is useful.

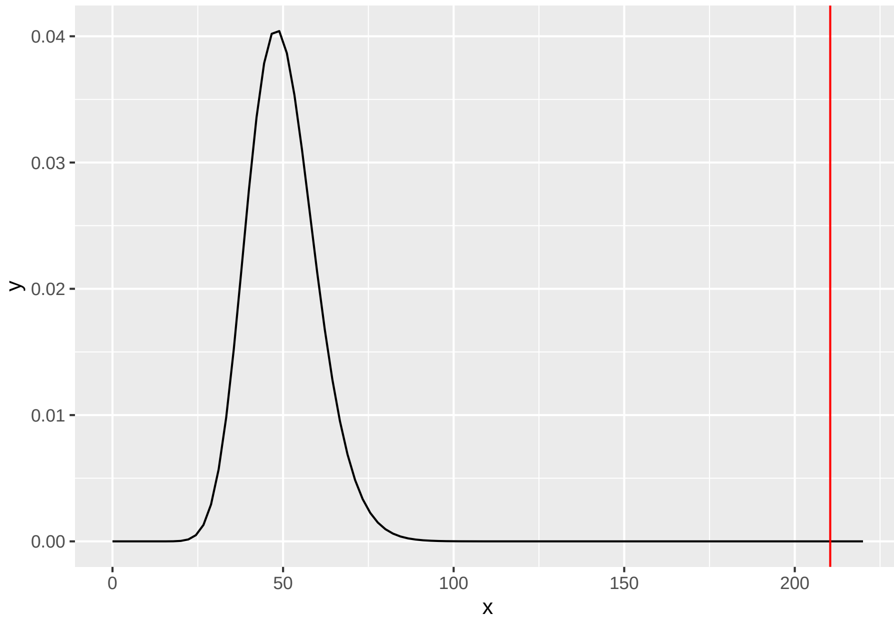
Is it as good as the saturated one? No (of course, it's not a perfect model).

```
pchisq(d1, df1, lower.tail = FALSE)
```

```
## [1] 1.44606e-21
```

Graphic representation:

```
tibble(
  x = seq(0, 220, length.out = 100),
  y = dchisq(x, df1)
) %>%
  ggplot() +
  geom_line(aes(x, y)) +
  geom_vline(aes(xintercept = d1), color = "red")
```



The plot tells us how far the fitted model is from being equal to the saturated one. It would be more useful though to observe the distance between the fitted and the null models.

> Exercise: plot the same graph for the deviance between the fit and the null model. Comment it.

8.5 Influence measures

For each data-point in the dataset, we can measure how that observation impacts the fit. The `influence.measures` function fits a new model with all observations but one, then compares the coefficients (in a sort of sensitivity analysis), and does this for every observation. The bigger is the difference in the coefficients, the higher is the influence of that datum. Note that the results depend from the parametrization of the model, (e.g. it changes when the reference level for tension is M or H).

```
infmeas <- influence.measures(breaks_glm)
str(infmeas)
```

```
## List of 3
##  $ infmat: num [1:54, 1:8] -0.366 -0.255 0.318 -0.395 0.678 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:54] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:8] "dfb.1_" "dfb.tnsM" "dfb.tnsH" "dfb.wolB" ...
##  $ is.inf: logi [1:54, 1:8] FALSE FALSE FALSE FALSE FALSE FALSE ...
```

```
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:54] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:8] "dfb.1_" "dfb.tnsM" "dfb.tnsH" "dfb.wolB" ...
##  $ call  : language glm(formula = breaks ~ tension + wool, family = "poisson", data = war
##  - attr(*, "class")= chr "infl"
```

```
infmeasmat <- infmeas$infmat
head(infmeasmat)
```

```
##       dfb.1_   dfb.tnsM   dfb.tnsH   dfb.wolB      dffit     cov.r     cook.d
## 1 -0.3663095  0.2043507  0.1925495  0.1866539 -0.3663095 1.0486985 0.12222528
## 2 -0.2551478  0.1423376  0.1341177  0.1300112 -0.2551478 1.1148176 0.06279697
## 3  0.3183334 -0.1775866 -0.1673310 -0.1622075  0.3183334 1.0795065 0.11798629
## 4 -0.3953939  0.2205759  0.2078377  0.2014739 -0.3953939 1.0285388 0.14014604
## 5  0.6776242 -0.3780219 -0.3561912 -0.3452850  0.6776242 0.7959885 0.54693078
## 6  0.2735269 -0.1525907 -0.1437786 -0.1393763  0.2735269 1.1052090 0.08642675
##          hat
## 1 0.08274043
## 2 0.08274043
## 3 0.08274043
## 4 0.08274043
## 5 0.08274043
## 6 0.08274043
```

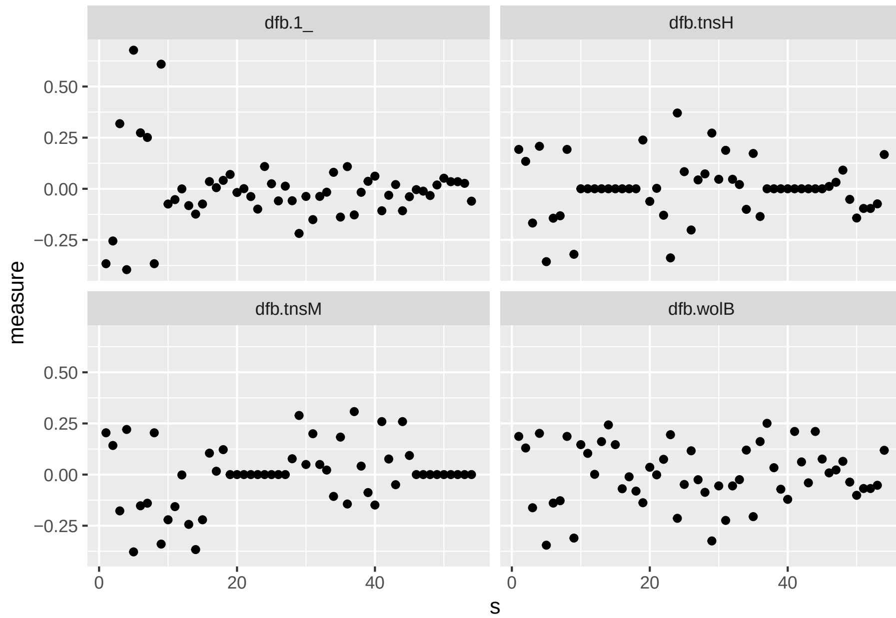
We can plot the observations influences:

```
summary(infmeas)
```

```
## Potentially influential observations of
##   glm(formula = breaks ~ tension + wool, family = "poisson", data = warpbreaks) :
## NONE
```

```
## numeric(0)
```

```
s <- list(s = 1:nrow(warpbreaks))
# we plot col 1, 2, 3 of influence measures
library(tidyr)
bind_cols(infmeasmat, s, warpbreaks) %>%
  gather(key = "infl_name", value = "measure", 1:4) %>%
  ggplot() +
  geom_point(aes(s, measure)) +
  facet_wrap(~infl_name)
```



8.6 Interaction

Let's add interaction.

```
# some other equivalent ways of writing breaks ~ tension*wool...
int_glm <- glm(breaks ~ tension + wool + tension:wool,
  data = warpbreaks, family = "poisson"
)
head(model.matrix(int_glm))
```

```
##   (Intercept) tensionM tensionH woolB tensionM:woolB tensionH:woolB
## 1           1        0        0     0              0              0
## 2           1        0        0     0              0              0
## 3           1        0        0     0              0              0
## 4           1        0        0     0              0              0
## 5           1        0        0     0              0              0
## 6           1        0        0     0              0              0
```

```
# as you can see from the design.matrix, it's the same model
int_glm2 <- glm(breaks ~ (tension + wool)^2,
  data = warpbreaks, family = "poisson"
)
head(model.matrix(int_glm2))
```

```
##   (Intercept) tensionM tensionH woolB tensionM:woolB tensionH:woolB
## 1           1        0        0     0              0              0
```

```
## 2           1        0        0     0              0              0
## 3           1        0        0     0              0              0
## 4           1        0        0     0              0              0
## 5           1        0        0     0              0              0
## 6           1        0        0     0              0              0
```

```r
summary(int_glm)
```

```
##
## Call:
## glm(formula = breaks ~ tension + wool + tension:wool, family = "poisson",
##     data = warpbreaks)
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     3.79674    0.04994  76.030  < 2e-16 ***
## tensionM       -0.61868    0.08440  -7.330 2.30e-13 ***
## tensionH       -0.59580    0.08378  -7.112 1.15e-12 ***
## woolB          -0.45663    0.08019  -5.694 1.24e-08 ***
## tensionM:woolB  0.63818    0.12215   5.224 1.75e-07 ***
## tensionH:woolB  0.18836    0.12990   1.450    0.147
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 297.37  on 53  degrees of freedom
## Residual deviance: 182.31  on 48  degrees of freedom
## AIC: 468.97
##
## Number of Fisher Scoring iterations: 4
```

> Exercise: draw an interaction plot and check if the data relate to an additive model or not. On the x-axis put the tension levels, on the y-axis the breaks. Draw two lines, one for wool A and one for wool B, passing through the mean number of breaks. How do you interpret the graph?

8.6.1 Testing interaction

Is a more complex model worth the additional parameters?

```r
AIC(breaks_glm, int_glm)
```

```
##            df      AIC
## breaks_glm  4 493.0560
## int_glm     6 468.9692
```

Yes, it is. And, again, we can test it against the null model.

```
d2 <- int_glm$deviance
df2 <- int_glm$df.residual
```

```
anova(breaks_glm, int_glm, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: breaks ~ tension + wool
## Model 2: breaks ~ tension + wool + tension:wool
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1        50     210.39
## 2        48     182.31  2   28.087 7.962e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
pchisq(d1 - d2, df1 - df2, lower.tail = FALSE) # Mfit - Mnull
```

```
## [1] 7.962292e-07
```

Chapter 9

Negative Binomial and Zero-Inflation

9.1 Dataset

It is a sample of 4,406 individuals, aged 66 and over, who were covered by Medicare in 1988. One of the variables the data provide is number of physician office visits. The dataset is provided by the AER package [Kleiber and Zeileis, 2008].

Our goal is to model the number of visits given the other attributes (chronic conditions, status, gender etc.). For sake of simplicity, we only select a subset of attributes which are considered to be relevant.

variable	description
visits	Number of physician office visits (integer outcome)
nvisits	
ovisits	
novisits	
emergency	
hospital	Number of hospital stays (integer)
health	Self-perceived health status (poor, average, excellent)
chronic	Number of chronic condition (integer)
adl	
region	
age	
afam	

variable	description
gender	Gender (female, male)
married	
school	Number of years of education (integer)
income	
employed	
insurance	Private insurance indicator (no, yes)
medicaid	

```
# load data from package
# install.package("AER")
library(AER)
```

```
## Loading required package: car

## Loading required package: carData

##
## Attaching package: 'car'

## The following object is masked from 'package:purrr':
##
##     some

## The following object is masked from 'package:dplyr':
##
##     recode

## Loading required package: lmtest

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: sandwich

## Loading required package: survival
```

```
data("NMES1988")
nmes <- NMES1988[, c(1, 6:8, 13, 15, 18)] # select variables of interest
summary(nmes)
```

```
##      visits          hospital           health        chronic
##   Min.   : 0.000   Min.   :0.000   poor     : 554   Min.   :0.000
```

```
##  1st Qu.: 1.000   1st Qu.:0.000   average  :3509   1st Qu.:1.000
##  Median : 4.000   Median :0.000   excellent: 343   Median :1.000
##  Mean   : 5.774   Mean   :0.296                    Mean   :1.542
##  3rd Qu.: 8.000   3rd Qu.:0.000                    3rd Qu.:2.000
##  Max.   :89.000   Max.   :8.000                    Max.   :8.000
##     gender          school       insurance
##  female:2628   Min.   : 0.00   no : 985
##  male  :1778   1st Qu.: 8.00   yes:3421
##                Median :11.00
##                Mean   :10.29
##                3rd Qu.:12.00
##                Max.   :18.00
```

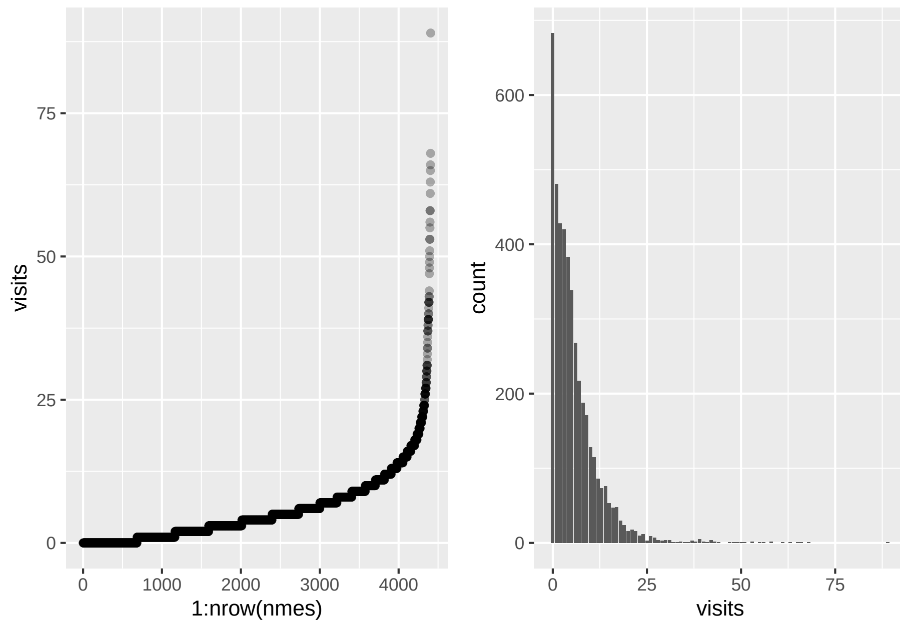
```
head(nmes)
```

```
##   visits hospital  health chronic gender school insurance
## 1      5        1 average       2   male      6       yes
## 2      1        0 average       2 female     10       yes
## 3     13        3    poor       4 female     10        no
## 4     16        1    poor       2   male      3       yes
## 5      3        0 average       2 female      6       yes
## 6     17        0    poor       5 female      7        no
```

9.2 EDA

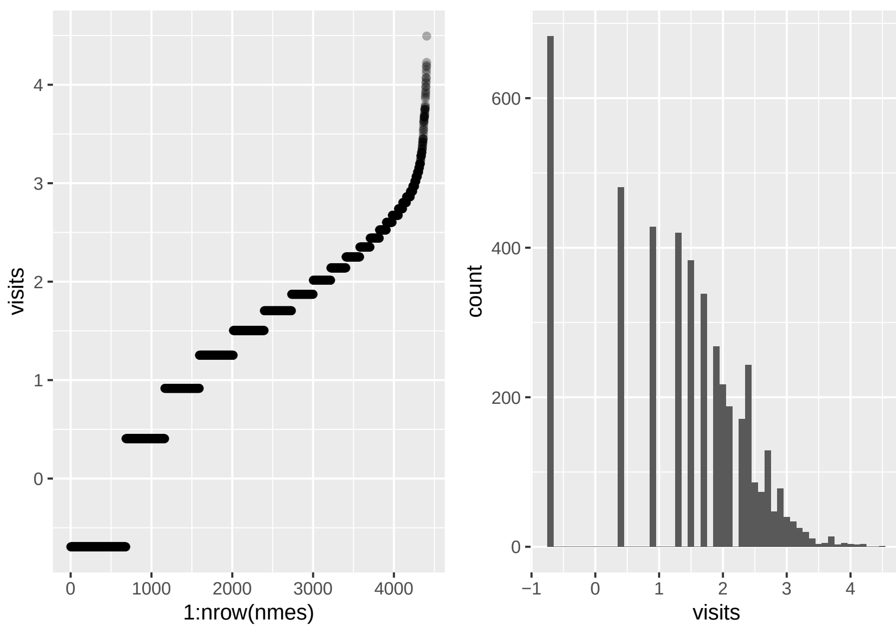
As always, let's start with some descriptive plots.

```
library(dplyr)
library(ggplot2)
library(ggpubr)

# analyse response
visits_scatter <- nmes %>%
  arrange(visits) %>%
  ggplot() +
  geom_point(aes(1:nrow(nmes), visits), alpha = .3)
visits_bar <- nmes %>%
  ggplot() +
  geom_bar(aes(visits))
ggarrange(visits_scatter, visits_bar, ncol = 2)
```



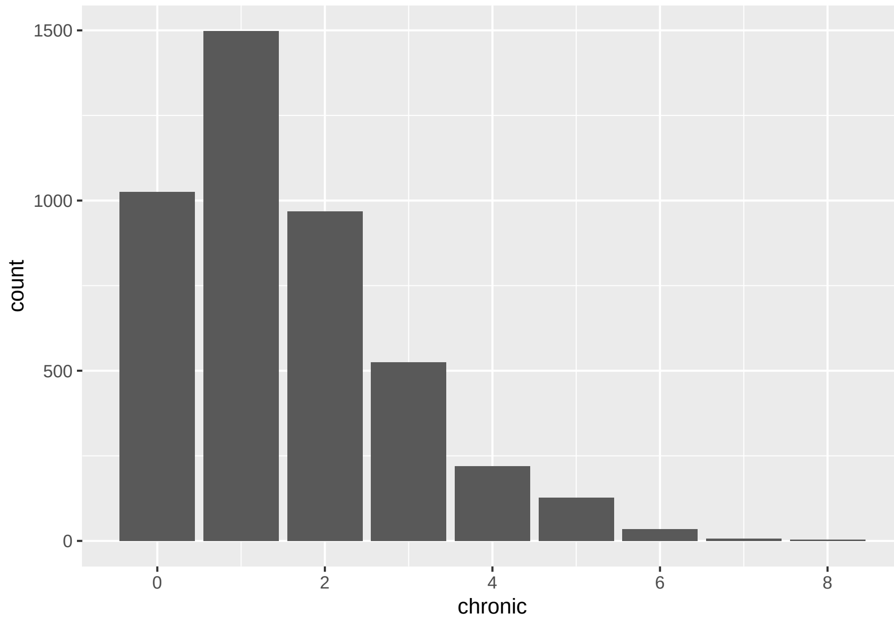
The response presents high variability, we can reduce this by taking the log.

```
logvisits_scatter <- nmes %>%
  mutate(visits = log(visits + .5)) %>%
  arrange(visits) %>%
  ggplot() +
  geom_point(aes(1:nrow(nmes), visits), alpha = .3)
logvisits_bar <- nmes %>%
  mutate(visits = log(visits + .5)) %>%
  ggplot() +
  geom_histogram(aes(visits), binwidth = .1)

ggarrange(logvisits_scatter, logvisits_bar, ncol = 2)
```



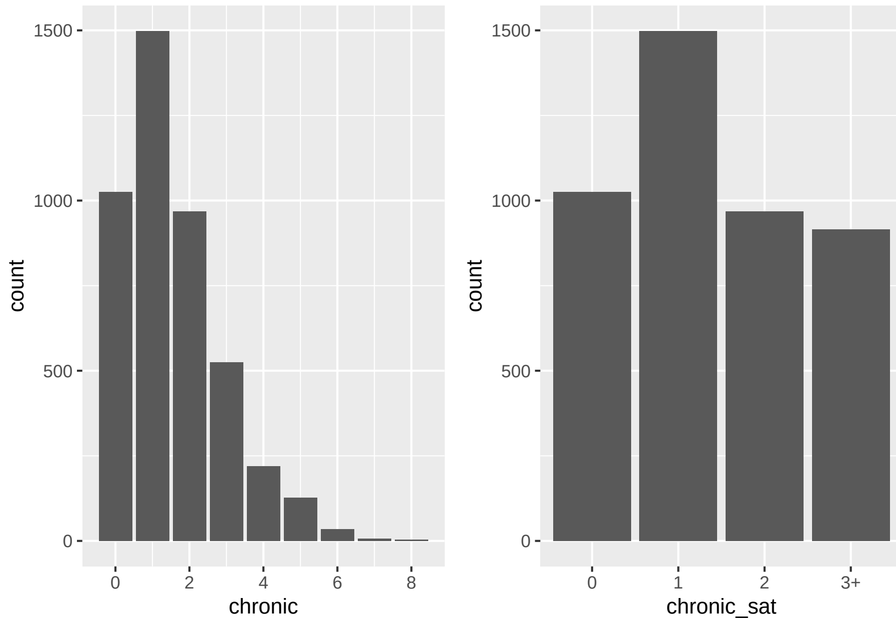
Now we analyse the relationship with the number of chronic diseases.

```
# saturate
sat <- function(x, sat_point) {
  x[x > sat_point] <- sat_point # saturate values above 3
  lev <- as.character(0:sat_point) # define levels
  lev[length(lev)] <- paste0(lev[length(lev)], "+")
  x <- as.factor(x) # switch from numeric to factor
  levels(x) <- lev
  return(x)
}

# analyse predictor "chronic"
unbalanced_plt <- nmes %>%
  ggplot() +
  geom_bar(aes(chronic))
unbalanced_plt
```



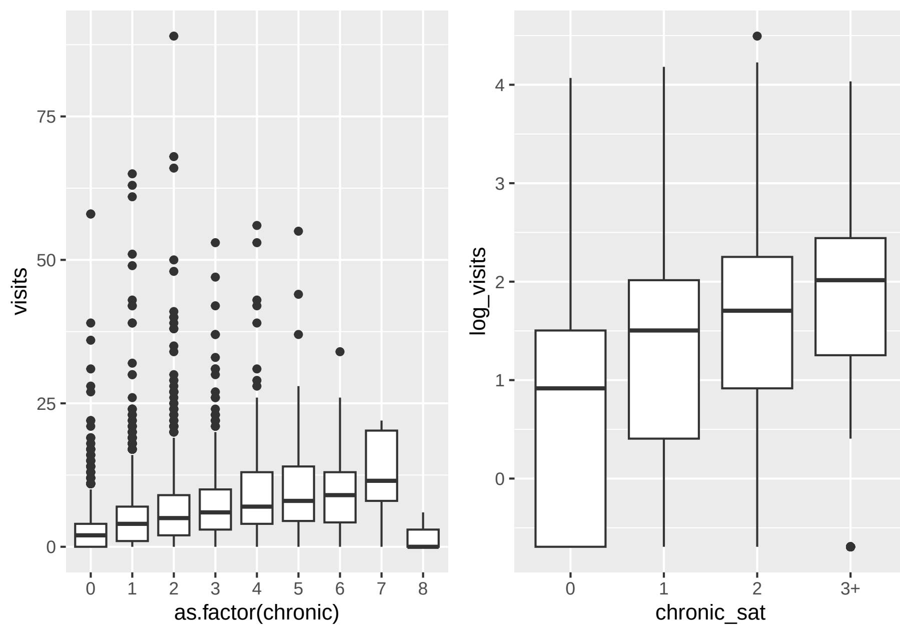
This variable seems to have too low support on values from 4 above. This will lead to low accuracy in the visit estimation when we observe a number of chronic diseases higher than 4. We may fix this issue by assuming that a patient having 4 or more diseases will get visited as frequently as a patient with 3 chronic diseases. This means that we set a saturation point at 3 and we transform the data accordingly.

```
balanced_plt <- nmes %>%
  mutate(chronic_sat = sat(chronic, 3)) %>%
  ggplot() +
  geom_bar(aes(chronic_sat))

ggarrange(unbalanced_plt, balanced_plt)
```



To summarize, here the two data transformations discussed so far (which we can compare to the original dataset). With a log-transform we fix the highly skewed distribution on the response variable, while with a saturation point on the `chronic` predictor, we create balanced classes.

```
# let's add two columns
nmes <- nmes %>%
  mutate(
    chronic_sat = sat(chronic, 3),
    log_visits = log(visits + .5)
  )

# bivariate analysis
biv <- nmes %>%
  ggplot() +
  geom_boxplot(aes(as.factor(chronic), visits))
tr_biv <- nmes %>%
  ggplot() +
  geom_boxplot(aes(chronic_sat, log_visits))

ggarrange(biv, tr_biv, ncol = 2)
```

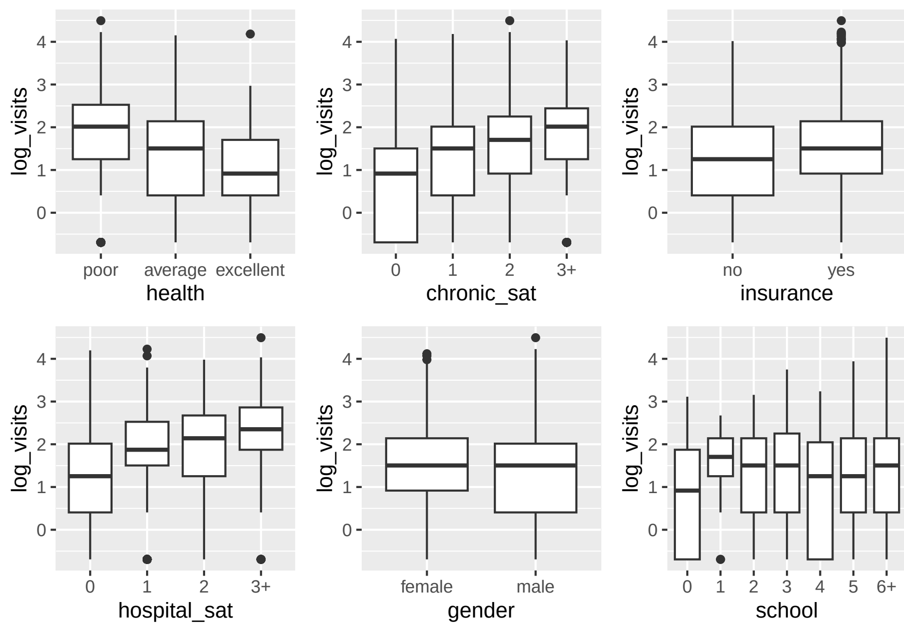


There are many less outliers and less variability among classes, which suggests that we can better capture its distribution

Here some other bi-variate plots which can be useful for further inspection of the available data.

```
library(tidyr)

plts <- list()

plts$health <- nmes %>%
  ggplot(aes(health, log_visits)) +
  geom_boxplot()
plts$chronic <- nmes %>%
  ggplot(aes(chronic_sat, log_visits)) +
  geom_boxplot()
plts$insurance <- nmes %>%
  ggplot(aes(insurance, log_visits)) +
  geom_boxplot()
plts$hospital <- nmes %>%
  mutate(hospital_sat = sat(hospital, 3)) %>%
  ggplot(aes(hospital_sat, log_visits)) +
  geom_boxplot()
plts$gender <- nmes %>%
  ggplot(aes(gender, log_visits)) +
  geom_boxplot()
```

```r
plts$school <- nmes %>%
  mutate(school = sat(school, 6)) %>%
  ggplot(aes(school, log_visits)) +
  geom_boxplot()

do.call(ggarrange, plts)
```



9.3 Negative binomial regression

The overdispersion of the data can be captured by a Negative Binomial model, which differs from the Poisson model in that the variance can be different than the mean. Therefore it can account for underdispersed and overdispersed count variates.

First we try with a simple Poisson regression

```r
# define a formula (select the relevant/interesting predictors)
fml <- visits ~ hospital + health + chronic_sat + gender + school + insurance

pois_model <- glm(
  formula = fml, family = poisson(link = "log"), # family object "poisson"
  data = nmes
)
summary(pois_model)
```

```
##
## Call:
## glm(formula = fml, family = poisson(link = "log"), data = nmes)
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)      0.848055   0.027418  30.930  < 2e-16 ***
## hospital         0.171588   0.005950  28.841  < 2e-16 ***
## healthpoor       0.277053   0.017401  15.922  < 2e-16 ***
## healthexcellent -0.312714   0.030443 -10.272  < 2e-16 ***
## chronic_sat1     0.361870   0.020591  17.574  < 2e-16 ***
## chronic_sat2     0.580241   0.021405  27.108  < 2e-16 ***
## chronic_sat3+    0.694679   0.021736  31.960  < 2e-16 ***
## gendermale      -0.104541   0.012963  -8.065 7.33e-16 ***
## school           0.025902   0.001842  14.062  < 2e-16 ***
## insuranceyes     0.191394   0.016902  11.323  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 26943  on 4405  degrees of freedom
## Residual deviance: 22928  on 4396  degrees of freedom
## AIC: 35723
##
## Number of Fisher Scoring iterations: 5
```

> Note: if we print out the coefficient, and we change the scale to match the counts, we see that, for instance, excellent health brings a decrease in the visits, while a number of chronic disease of three or higher, dramatically increases the visits count.

```
coef(pois_model)
```

```
##     (Intercept)        hospital      healthpoor healthexcellent    chronic_sat1
##      0.84805502      0.17158798      0.27705323     -0.31271426      0.36186975
##    chronic_sat2   chronic_sat3+      gendermale          school    insuranceyes
##      0.58024070      0.69467856     -0.10454072      0.02590154      0.19139428
```

```
exp(coef(pois_model))
```

```
##     (Intercept)        hospital      healthpoor healthexcellent    chronic_sat1
##       2.3351007       1.1871886       1.3192366       0.7314589       1.4360119
##    chronic_sat2   chronic_sat3+      gendermale          school    insuranceyes
##       1.7864684       2.0030651       0.9007381       1.0262399       1.2109368
```

Then we compare the results with a regression fit on a GLM with Negative Binomial family.

```r
# fit the equivalent NB model
# check
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```r
nb_model <- glm.nb(formula = fml, data = nmes)
summary(nb_model)
```

```
##
## Call:
## glm.nb(formula = fml, data = nmes, init.theta = 1.218804554,
##     link = log)
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)      0.792055   0.059558  13.299  < 2e-16 ***
## hospital         0.220185   0.020042  10.986  < 2e-16 ***
## healthpoor       0.318920   0.047833   6.667 2.60e-11 ***
## healthexcellent -0.317113   0.061183  -5.183 2.18e-07 ***
## chronic_sat1     0.372721   0.042733   8.722  < 2e-16 ***
## chronic_sat2     0.575107   0.047135  12.201  < 2e-16 ***
## chronic_sat3+    0.721231   0.049283  14.635  < 2e-16 ***
## gendermale      -0.118742   0.031166  -3.810 0.000139 ***
## school           0.027295   0.004381   6.230 4.67e-10 ***
## insuranceyes     0.207695   0.039357   5.277 1.31e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(1.2188) family taken to be 1)
##
##     Null deviance: 5782.5  on 4405  degrees of freedom
## Residual deviance: 5044.7  on 4396  degrees of freedom
## AIC: 24330
##
## Number of Fisher Scoring iterations: 1
##
##
##               Theta:  1.2188
##           Std. Err.:  0.0340
##
##  2 x log-likelihood:  -24308.2680
```

Among the fit information we can see that the `glm.nb` function estimates the dispersion parameter of the Negative Binomial. Keep in mind that there are several parametrization of this distribution, one of which consists of, indeed, mean μ and dispersion r i.e. $NB(\mu, r)$ such that

$$\sigma^2 = \mu + \frac{\mu^2}{r}\,, \qquad p = \frac{m}{\sigma^2}$$

```
# coefficients again
exp(coef(nb_model))
```

```
##     (Intercept)        hospital      healthpoor healthexcellent    chronic_sat1
##       2.2079292       1.2463073       1.3756414       0.7282484       1.4516787
##    chronic_sat2   chronic_sat3+      gendermale          school    insuranceyes
##       1.7773208       2.0569629       0.8880371       1.0276705       1.2308380
```

Let's compare the coefficients and the confidence intervals:

```
rbind(exp(coef(pois_model)), exp(coef(nb_model)))
```

```
##      (Intercept) hospital healthpoor healthexcellent chronic_sat1 chronic_sat2
## [1,]    2.335101 1.187189   1.319237       0.7314589     1.436012     1.786468
## [2,]    2.207929 1.246307   1.375641       0.7282484     1.451679     1.777321
##      chronic_sat3+ gendermale   school insuranceyes
## [1,]      2.003065  0.9007381 1.026240     1.210937
## [2,]      2.056963  0.8880371 1.027671     1.230838
```

```
cbind(confint.default(pois_model), confint.default(nb_model))
```

```
##                      2.5 %      97.5 %       2.5 %      97.5 %
## (Intercept)      0.7943160  0.90179405  0.67532364  0.90878650
## hospital         0.1599271  0.18324884  0.18090387  0.25946612
## healthpoor       0.2429475  0.31115895  0.22516928  0.41267092
## healthexcellent -0.3723819 -0.25304665 -0.43703008 -0.19719596
## chronic_sat1     0.3215120  0.40222748  0.28896513  0.45647604
## chronic_sat2     0.5382882  0.62219320  0.48272398  0.66749012
## chronic_sat3+    0.6520769  0.73728020  0.62463832  0.81782282
## gendermale      -0.1299469 -0.07913454 -0.17982625 -0.05765723
## school           0.0222914  0.02951169  0.01870754  0.03588172
## insuranceyes     0.1582661  0.22452241  0.13055767  0.28483287
```

```
cbind(
  confint.default(pois_model)[, 2] - confint.default(pois_model)[, 1],
  confint.default(nb_model)[, 2] - confint.default(nb_model)[, 1]
)
```

```
##                        [,1]       [,2]
## (Intercept)     0.107478074 0.23346286
## hospital        0.023321722 0.07856225
```

```
## healthpoor      0.068211444 0.18750164
## healthexcellent 0.119335218 0.23983412
## chronic_sat1    0.080715454 0.16751091
## chronic_sat2    0.083904993 0.18476614
## chronic_sat3+   0.085203273 0.19318450
## gendermale      0.050812349 0.12216902
## school          0.007220284 0.01717418
## insuranceyes    0.066256263 0.15427520
```

The confidence intervals are wider, which is an effect of the Negative Binomial letting more uncertainty in the model.

9.4 Zero-Inflation

With a regression fit, we only obtain the means of the Poisson, or Negative Binomial, distributions for each observations. These are the *fitted values*. However, many observations have response variable counting 0 visits. How many zeros does our model predicts?

```r
mu <- predict(pois_model, type = "response") # get the poisson mean
expected_zero_count <- sum(dpois(x = 0, lambda = mu)) # sum_i (1(yi == 0) * p(yi == 0))
round(expected_zero_count)
```

```
## [1] 60
```

How many are actually 0? Many more...

```r
# actual 0 visits
sum(nmes$visits == 0)
```

```
## [1] 683
```

For this reason we introduce a composite model called Zero-Inflated Poisson: it's a mixture between a Poisson and a discrete distribution over zero.

$$P(Y = y) = \pi 1(y = 0) + (1 - \pi)\frac{\lambda^y e^{-\lambda}}{y!}$$

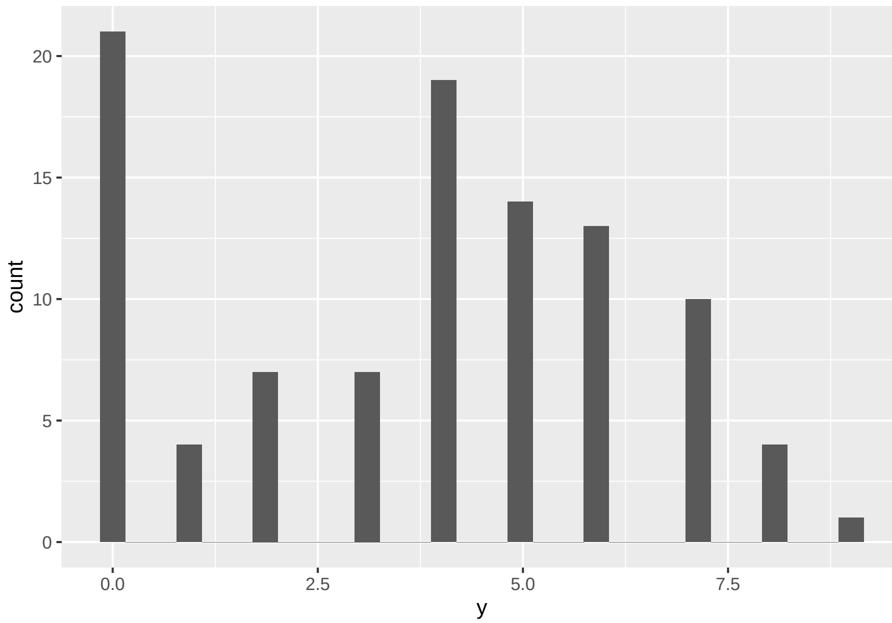
Let's have a look at the generative model and the distribution of ZIP variates. It can be seen as a two-steps process:

1. Sample a Bernoulli variable which states whether the observation is zero or not-zero
2. If it is not zero, then sample a Poisson variable with a given mean

```r
n <- 100
pp <- .3 # probability of zero event
ll <- 5
zi_samples <- rbinom(n, 1, 1 - pp)
```

```
zi_samples[zi_samples == 1] <- rpois(sum(zi_samples), lambda = 11) # sample poisson
tibble(y = zi_samples) %>%
  ggplot() +
  geom_histogram(aes(y))
```

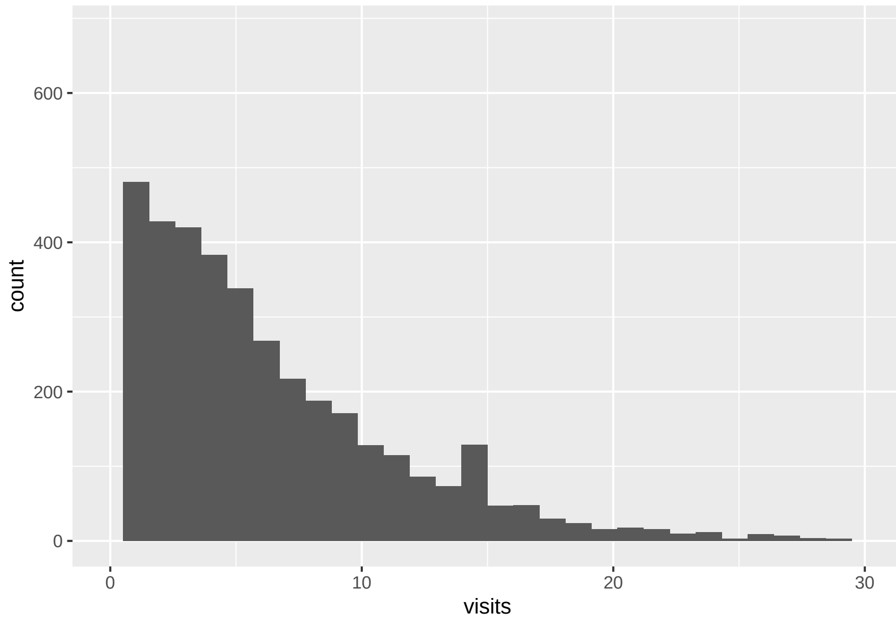
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



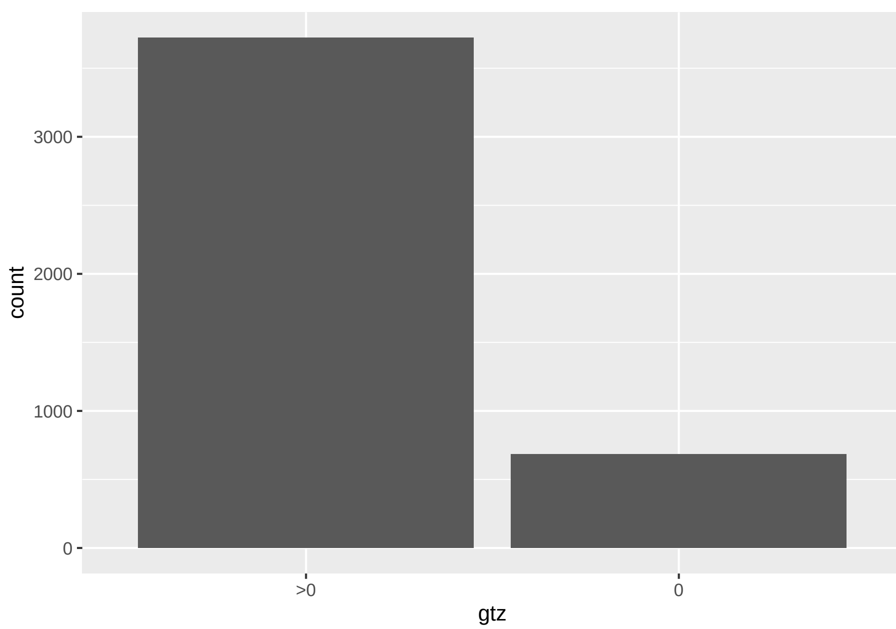
With our data, we do not observe something really like a zero inflation model, but still, with a ZI model, we can have more insight on the presence of many zeros.

```
nmes %>%
  ggplot() +
  geom_histogram(aes(visits)) +
  xlim(0, 30)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 47 rows containing non-finite outside the scale range
## (`stat_bin()`).

## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_bar()`).
```




```
nmes %>%
  mutate(gtz = as.factor(ifelse(visits > 0, ">0", "0"))) %>%
  ggplot() +
  geom_bar(aes(gtz))
```



Let's try to fit a zero inflation model.

Notice how the `zeroinfl` function infers two models: the count (pois) and the zero model (logistic regression).

```r
library(pscl)
```

```
## Classes and Methods for R originally developed in the
## Political Science Computational Laboratory
## Department of Political Science
## Stanford University (2002-2015),
## by and under the direction of Simon Jackman.
## hurdle and zeroinfl functions by Achim Zeileis.
```

```r
zip_model <- zeroinfl(formula = fml, data = nmes)
summary(zip_model)
```

```
##
## Call:
## zeroinfl(formula = fml, data = nmes)
##
## Pearson residuals:
##     Min      1Q  Median      3Q     Max
## -4.5758 -1.1488 -0.4766  0.5484 24.6115
##
## Count model coefficients (poisson with log link):
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)      1.307611   0.028121  46.499  < 2e-16 ***
## hospital         0.162911   0.006033  27.004  < 2e-16 ***
## healthpoor       0.278539   0.017354  16.051  < 2e-16 ***
## healthexcellent -0.283549   0.031288  -9.063  < 2e-16 ***
## chronic_sat1     0.212773   0.020937  10.163  < 2e-16 ***
## chronic_sat2     0.361709   0.021714  16.658  < 2e-16 ***
## chronic_sat3+    0.440948   0.021973  20.067  < 2e-16 ***
## gendermale      -0.057361   0.013071  -4.388 1.14e-05 ***
## school           0.019133   0.001873  10.215  < 2e-16 ***
## insuranceyes     0.077189   0.017174   4.494 6.98e-06 ***
##
## Zero-inflation model coefficients (binomial with logit link):
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)      0.04724    0.15010   0.315  0.75294
## hospital        -0.30074    0.09149  -3.287  0.00101 **
## healthpoor       0.01370    0.16175   0.085  0.93249
## healthexcellent  0.18472    0.15263   1.210  0.22619
## chronic_sat1    -0.74134    0.10483  -7.072 1.53e-12 ***
## chronic_sat2    -1.31875    0.13834  -9.533  < 2e-16 ***
## chronic_sat3+   -1.86784    0.17077 -10.938  < 2e-16 ***
```

```
## gendermale        0.40582    0.08994   4.512 6.41e-06 ***
## school           -0.05698    0.01232  -4.624 3.77e-06 ***
## insuranceyes     -0.75192    0.10316  -7.289 3.13e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Number of iterations in BFGS optimization: 27
## Log-likelihood: -1.611e+04 on 20 Df
```

```
round(sum(predict(zip_model, type = "zero"))) # better captures the zero
```

```
## [1] 669
```

```
# can account also for overdispersion
zinb_model <- zeroinfl(formula = fml, dist = "negbin", data = nmes)
summary(zinb_model)
```

```
##
## Call:
## zeroinfl(formula = fml, data = nmes, dist = "negbin")
##
## Pearson residuals:
##     Min      1Q  Median      3Q     Max
## -1.1938 -0.7112 -0.2775  0.3293 17.1983
##
## Count model coefficients (negbin with log link):
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)       1.096144   0.063810  17.178  < 2e-16 ***
## hospital          0.202671   0.020602   9.837  < 2e-16 ***
## healthpoor        0.303138   0.046057   6.582 4.65e-11 ***
## healthexcellent -0.304768   0.063342  -4.812 1.50e-06 ***
## chronic_sat1      0.260039   0.045197   5.753 8.75e-09 ***
## chronic_sat2      0.419531   0.048339   8.679  < 2e-16 ***
## chronic_sat3+     0.542364   0.049509  10.955  < 2e-16 ***
## gendermale       -0.072537   0.031212  -2.324   0.0201 *
## school            0.022117   0.004415   5.010 5.44e-07 ***
## insuranceyes      0.106747   0.042311   2.523   0.0116 *
## Log(theta)        0.401063   0.036208  11.077  < 2e-16 ***
##
## Zero-inflation model coefficients (binomial with logit link):
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -0.21623    0.29103  -0.743 0.457493
## hospital         -0.82495    0.52315  -1.577 0.114820
## healthpoor        0.14127    0.41427   0.341 0.733091
## healthexcellent  0.05502    0.33699   0.163 0.870316
## chronic_sat1     -1.06685    0.22995  -4.639 3.49e-06 ***
## chronic_sat2     -2.22231    0.45963  -4.835 1.33e-06 ***
```

```
## chronic_sat3+   -3.42831    0.96764  -3.543 0.000396 ***
## gendermale       0.70380    0.20796   3.384 0.000714 ***
## school          -0.07604    0.02789  -2.726 0.006412 **
## insuranceyes    -1.25027    0.23645  -5.288 1.24e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Theta = 1.4934
## Number of iterations in BFGS optimization: 36
## Log-likelihood: -1.209e+04 on 21 Df
```

```
exp_coeff <- exp(coef(zip_model))
exp_coeff <- matrix(exp_coeff, ncol = 2)
colnames(exp_coeff) <- c("count", "zero")
exp_coeff
```

```
##           count      zero
##  [1,] 3.6973299 1.0483782
##  [2,] 1.1769320 0.7402724
##  [3,] 1.3211986 1.0137962
##  [4,] 0.7531066 1.2028785
##  [5,] 1.2371035 0.4764769
##  [6,] 1.4357804 0.2674684
##  [7,] 1.5541806 0.1544569
##  [8,] 0.9442534 1.5005307
##  [9,] 1.0193167 0.9446120
## [10,] 1.0802458 0.4714595
```

We can also set different models for the two parts

E.g. from the previous summary it seems that health does not affect the visits count. Maybe we can remove it:

```
zinb_2model <- zeroinfl(
  formula = visits ~
    hospital + health + chronic_sat + gender + school + insurance | # count
      hospital + chronic_sat + gender + school + insurance, # zero
  dist = "negbin",
  data = nmes
)
summary(zinb_2model)
```

```
##
## Call:
## zeroinfl(formula = visits ~ hospital + health + chronic_sat + gender +
##     school + insurance | hospital + chronic_sat + gender + school + insurance,
##     data = nmes, dist = "negbin")
##
```

```
## Pearson residuals:
##     Min      1Q  Median      3Q     Max 
## -1.1930 -0.7129 -0.2774  0.3325 17.2308 
## 
## Count model coefficients (negbin with log link):
##                  Estimate Std. Error z value Pr(>|z|)    
## (Intercept)      1.095975   0.063712  17.202  < 2e-16 ***
## hospital         0.203030   0.020520   9.894  < 2e-16 ***
## healthpoor       0.299896   0.045025   6.661 2.73e-11 ***
## healthexcellent -0.307842   0.060352  -5.101 3.38e-07 ***
## chronic_sat1     0.261129   0.045084   5.792 6.95e-09 ***
## chronic_sat2     0.420119   0.048342   8.691  < 2e-16 ***
## chronic_sat3+    0.542827   0.049521  10.962  < 2e-16 ***
## gendermale      -0.073013   0.031222  -2.339   0.0194 *  
## school           0.022174   0.004399   5.041 4.64e-07 ***
## insuranceyes     0.105967   0.042156   2.514   0.0119 *  
## Log(theta)       0.399783   0.036029  11.096  < 2e-16 ***
## 
## Zero-inflation model coefficients (binomial with logit link):
##               Estimate Std. Error z value Pr(>|z|)    
## (Intercept)   -0.20007    0.28703  -0.697 0.485788    
## hospital      -0.81416    0.49144  -1.657 0.097584 .  
## chronic_sat1  -1.05197    0.22352  -4.706 2.52e-06 ***
## chronic_sat2  -2.21569    0.46239  -4.792 1.65e-06 ***
## chronic_sat3+ -3.48962    1.06920  -3.264 0.001099 ** 
## gendermale     0.69879    0.20783   3.362 0.000773 ***
## school        -0.07574    0.02732  -2.772 0.005564 ** 
## insuranceyes  -1.26793    0.23136  -5.480 4.25e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Theta = 1.4915 
## Number of iterations in BFGS optimization: 33 
## Log-likelihood: -1.209e+04 on 19 Df
```

And since the regression model is composed of two models, also the prediction can be split into the two parts as such:

```r
# two models predictions and combined
predict(zip_model, type = "zero")[1:5]
```

```
##          1          2          3          4          5 
## 0.09447018 0.06957463 0.03630164 0.11149199 0.08585591
```

```r
predict(zip_model, type = "count")[1:5]
```

```
##         1         2         3         4         5 
##  7.148150  6.943690 14.986611  8.917319  6.432114
```

```r
predict(zip_model, type = "response")[1:5]
```

```
##         1         2         3         4         5
##  6.472863  6.460585 14.442573  7.923110  5.879879
```

And finally we can test the models one against the other, comparing the likelihood or using the AIC score.

```r
library(lmtest)
# likelihood test
lmtest::lrtest(zip_model, zinb_2model)
```

```
## Likelihood ratio test
##
## Model 1: visits ~ hospital + health + chronic_sat + gender + school +
##     insurance
## Model 2: visits ~ hospital + health + chronic_sat + gender + school +
##     insurance | hospital + chronic_sat + gender + school + insurance
##   #Df LogLik Df Chisq Pr(>Chisq)
## 1  20 -16107
## 2  19 -12085 -1  8044  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# p-value tells whether the logLikelihood difference is significant

# AIC for model selection based on complexity/performance tradeoff
AIC(pois_model, nb_model, zip_model, zinb_2model)
```

```
##             df      AIC
## pois_model  10 35723.15
## nb_model    11 24330.27
## zip_model   20 32254.65
## zinb_2model 19 24208.69
```

9.4.1 Hurdle v. ZI

In general, zeros can come both from the zero model and the count model. Sometimes we might want to model a zero as a separate event. The Hurdle model in fact, alternatively to the ZI model, makes a clear distinction between counts that are zero and counts that are 1 or more. Here is the Hurdle-Poisson model for instance:

$$P(Y = y) = \pi 1(y = 0) + (1 - \pi)\frac{\lambda^y e^{-\lambda}}{y!} 1(y > 0)$$

This is particularly useful when the data consist of large counts on average, but for some reasons (e.g. reading errors) many values are 0 instead. It probably doesn't make sense, in those cases, to account for the probability of that zero being drawn from the same Poisson of that of all the other larger counts.

In R, this can be done by using the `hurdle` function from the same library.

```
hurdle_model <- hurdle(formula = fml, data = nmes)
summary(hurdle_model)
```

```
##
## Call:
## hurdle(formula = fml, data = nmes)
##
## Pearson residuals:
##     Min      1Q  Median      3Q     Max
## -4.5965 -1.1507 -0.4768  0.5482 24.5851
##
## Count model coefficients (truncated poisson with log link):
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)       1.308658   0.028104  46.565  < 2e-16 ***
## hospital          0.162841   0.006034  26.988  < 2e-16 ***
## healthpoor        0.278497   0.017355  16.047  < 2e-16 ***
## healthexcellent -0.282804   0.031274  -9.043  < 2e-16 ***
## chronic_sat1      0.212190   0.020935  10.136  < 2e-16 ***
## chronic_sat2      0.361344   0.021708  16.646  < 2e-16 ***
## chronic_sat3+     0.440479   0.021973  20.046  < 2e-16 ***
## gendermale       -0.057295   0.013071  -4.383 1.17e-05 ***
## school            0.019050   0.001871  10.180  < 2e-16 ***
## insuranceyes      0.077519   0.017166   4.516 6.31e-06 ***
## Zero hurdle model coefficients (binomial with logit link):
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -0.0981863  0.1468254  -0.669 0.503669
## hospital          0.3107238  0.0913541   3.401 0.000671 ***
## healthpoor       -0.0001956  0.1611027  -0.001 0.999031
## healthexcellent -0.2419271  0.1442703  -1.677 0.093562 .
## chronic_sat1      0.7585597  0.1024341   7.405 1.31e-13 ***
## chronic_sat2      1.3396653  0.1359419   9.855  < 2e-16 ***
## chronic_sat3+     1.8896854  0.1686629  11.204  < 2e-16 ***
## gendermale       -0.4052651  0.0881205  -4.599 4.25e-06 ***
## school            0.0589871  0.0120460   4.897 9.74e-07 ***
## insuranceyes      0.7449223  0.1012575   7.357 1.88e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Number of iterations in BFGS optimization: 16
## Log-likelihood: -1.611e+04 on 20 Df
```

```
z_pred <- predict(hurdle_model, type = "zero")[1:5]
c_pred <- predict(hurdle_model, type = "count")[1:5]
predict(hurdle_model, type = "response")[1:5]
```

```
##         1         2         3         4         5
##  6.472374  6.460350 14.453157  7.923691  5.879604
```

```
# here the composite prediction is merely the product of the two
# models predictions
z_pred * c_pred
```

```
##         1         2         3         4         5
##  6.472374  6.460350 14.453157  7.923691  5.879604
```

Chapter 10

Three JAGS examples

10.1 Rats - Simple linear regression

We use the data from [Gelfand et al., 1990] and select the 9th observation which features the weight at 5 different time points of a single rat.

We estimate the intercept and the coefficient in a Bayesian framework using JAGS, then validate our result with the traditional `lm`. This example is taken from [Lunn et al., 2013]

```
set.seed(101)
# load data
# install.packages("R2MLwiN")

data(rats, package = "R2MLwiN")
y <- unlist(rats[9, 1:5])
x <- c(8, 15, 22, 29, 36)
```

```
library(R2jags)
```

```
## Loading required package: rjags
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.3.2
```

```
## Loaded modules: basemod,bugs
```

```
##
## Attaching package: 'R2jags'
```

```
## The following object is masked from 'package:coda':
##
##     traceplot
```

```r
model_code <- "
model {
  for (i in 1:5) {
    y[i] ~ dnorm(mu[i], tau)
    mu[i] <- alpha + beta*x[i]
  }

  alpha ~ dnorm(0, 10^-5)
  beta ~ dnorm(0, 10^-5)
  tau ~ dgamma(0.0001, 0.0001)
  sigma2 <- 1 / tau
}
"
model_data <- list(y = y, x = x)
model_params <- c("alpha", "beta", "sigma2")
model_run <- jags(
  data = model_data,
  parameters.to.save = model_params,
  model.file = textConnection(model_code),
  n.chains = 2, n.burnin = 500, n.iter = 5000
)
```

```
## module glm loaded
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 5
##    Unobserved stochastic nodes: 3
##    Total graph size: 31
##
## Initializing model
```

```r
model_run$BUGSoutput
```

```
## Inference for Bugs model at "4", fit using jags,
##  2 chains, each with 5000 iterations (first 500 discarded), n.thin = 4
##  n.sims = 2250 iterations saved
##           mean     sd 2.5%   25%   50%   75%  97.5% Rhat n.eff
## alpha    123.5   19.8 85.6 114.3 123.7 132.3  160.9    1  1000
## beta       7.3    0.8  5.8   7.0   7.3   7.7    8.9    1  2200
## deviance  39.9    3.8 35.8  37.2  38.9  41.6   49.5    1  2200
## sigma2   450.8 3223.0 40.2  87.5 153.9 294.9 2028.3    1  2200
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
```

```
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 7.2 and DIC = 47.2
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```r
library(tibble)

lmfit <- lm(y ~ x, data = tibble(x = x, y = y))
summary(lmfit)
```

```
##
## Call:
## lm(formula = y ~ x, data = tibble(x = x, y = y))
##
## Residuals:
##     1     2     3     4     5
##  -5.4   2.4   0.2  14.0 -11.2
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 123.8857    11.8788   10.43 0.001882 **
## x             7.3143     0.4924   14.86 0.000662 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.9 on 3 degrees of freedom
## Multiple R-squared:  0.9866, Adjusted R-squared:  0.9821
## F-statistic: 220.7 on 1 and 3 DF,  p-value: 0.000662
```

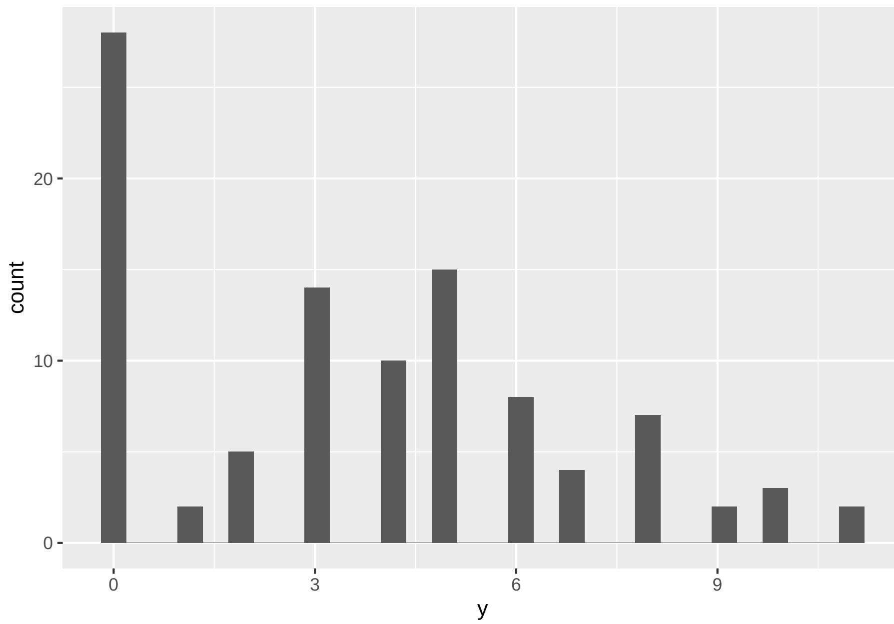
We obtain same values for α (intercept) and β (slope).

10.2 ZIP model

We generate some synthetic data according to a set of pre-defined parameters (p, λ).

```r
library(ggplot2)

n <- 100
pp <- .3 # probability of zero event
ll <- 5
zi_sample <- rbinom(n, 1, 1 - pp)
zi_sample[zi_sample == 1] <- rpois(sum(zi_sample), lambda = ll) # sample poisson

tibble(y = zi_sample) %>%
  ggplot() +
  geom_histogram(aes(y))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
model_code <- "
model {
  for (i in 1:N) {
    y[i] ~ dpois(m[i])
    m[i] <- group[i] * mu
    group[i] ~ dbern(p)
  }

  p ~ dunif(0, 1) # probability of y being drawn from a Poisson
  mu ~ dgamma(0.5, 0.0001)
}
"

model_data <- list(y = zi_sample, N = n)
model_params <- c("mu", "p")
model_run <- jags(
  data = model_data,
  parameters.to.save = model_params,
  model.file = textConnection(model_code),
  n.chains = 2, n.burnin = 500, n.iter = 2000
)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 102
##    Total graph size: 307
##
## Initializing model
```

```r
model_run$BUGSoutput
```

```
## Inference for Bugs model at "5", fit using jags,
##  2 chains, each with 2000 iterations (first 500 discarded)
##  n.sims = 3000 iterations saved
##           mean  sd  2.5%   25%   50%   75% 97.5% Rhat n.eff
## deviance 329.0 6.9 323.5 323.8 324.9 334.0 345.2    1  3000
## mu         5.1 0.3   4.6   5.0   5.1   5.3   5.7    1  1600
## p          0.7 0.0   0.6   0.7   0.7   0.8   0.8    1  1000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 24.1 and DIC = 353.1
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Check that: - the true values fall inside the 95% CI - Rhat is close to 1 (chain convergence)

10.3 Gaussian Mixture Model (GMM)

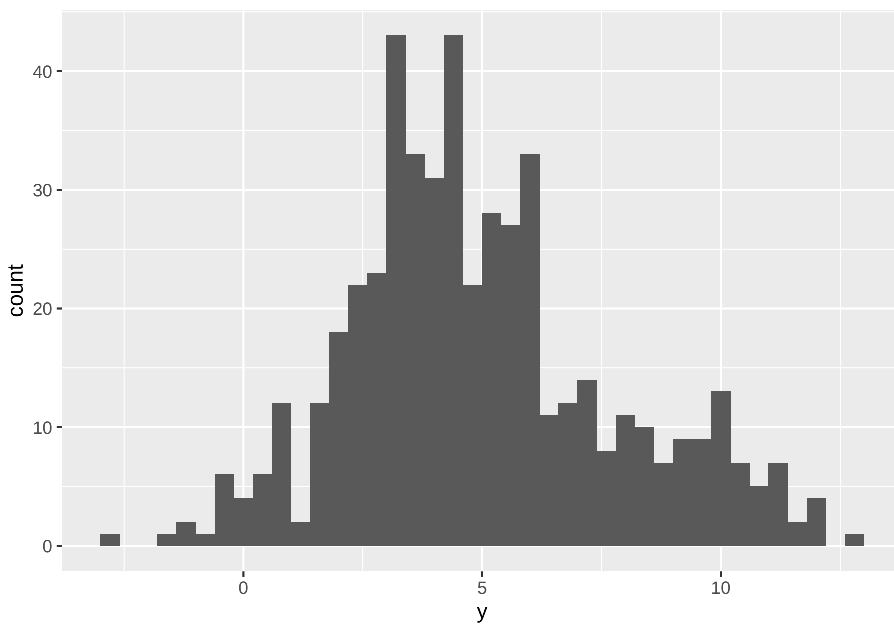
GMMs are used for clustering data, i.e. group observations which are similar and come from a Gaussian with same mean. It is called *mixture* in that every observation comes from a Gaussian with a certain mean, and the mean depends in turn on the component/cluster to which it belongs. Therefore the mean is a random variable selected among multiple means (Categorical distributed).

Other mixture models are the ZIP model (mixture of a Poisson and a Bernoulli distribution) and the Negative Binomial (mixture of a Poisson and a Gamma - details).

We generate a sample from a GMM with just two components, arbitrarily defining the true parameters.

```r
# synthetic dataset
n <- 500
group_prob <- .2
```

```r
means <- c(4., 9.)
stdev <- 2 # same sd for simplicity

# sample the component to which the observation
mixt_groups <- rbinom(n, 1, group_prob) # bernoulli trials
# sample the Gaussian variable setting the mean
# equal to means[1] or means[2] depending on the
# group to which each observation belongs
mixt_sample <- rnorm(n,
  mean = unlist(lapply(mixt_groups, FUN = function(g) means[g + 1])),
  sd = stdev
)

# plot the data
tibble(y = mixt_sample) %>%
  ggplot() +
  geom_histogram(aes(y), binwidth = 0.4)
```




```r
model_code <- "
model {
  for(i in 1:N) {
    y[i] ~ dnorm(mu[i] , tau)
    mu[i] <- mu_clust[clust[i] + 1]
    clust[i] ~ dbern(gp)
  }
```

```
  # priors
  gp ~ dbeta(1, 1) # uniform prior
  tau ~ dgamma(0.01, 0.01)
  sigma <- sqrt(1/tau)

  mu_clust_raw[1] ~ dnorm(0, 10^-2)
  mu_clust_raw[2] ~ dnorm(0, 10^-2)

  mu_clust <- sort(mu_clust_raw) # ensure order to prevent label switch
}
"

model_data <- list(y = mixt_sample, N = n)
# save the parameters and, optionally, the
# `clust` variable representing the labeling
# of each observation (clust[i] = 1 if y[i] is found
# to belong to the second cluster, 0 otherwise)
model_params <- c(
  "mu_clust", "gp", "sigma"
  # , "clust"
)
model_run <- jags(
  data = model_data,
  parameters.to.save = model_params,
  model.file = textConnection(model_code),
  n.chains = 2, n.burnin = 1000, n.iter = 5000
)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 500
##    Unobserved stochastic nodes: 504
##    Total graph size: 2018
##
## Initializing model
```

```
model_run$BUGSoutput
```

```
## Inference for Bugs model at "6", fit using jags,
##  2 chains, each with 5000 iterations (first 1000 discarded), n.thin = 4
##  n.sims = 2000 iterations saved
##               mean   sd   2.5%    25%    50%    75%  97.5% Rhat n.eff
## deviance    2066.9 33.3 2009.2 2043.6 2062.9 2087.4 2138.1    1  2000
## gp             0.2  0.0    0.2    0.2    0.2    0.2    0.3    1  1700
```

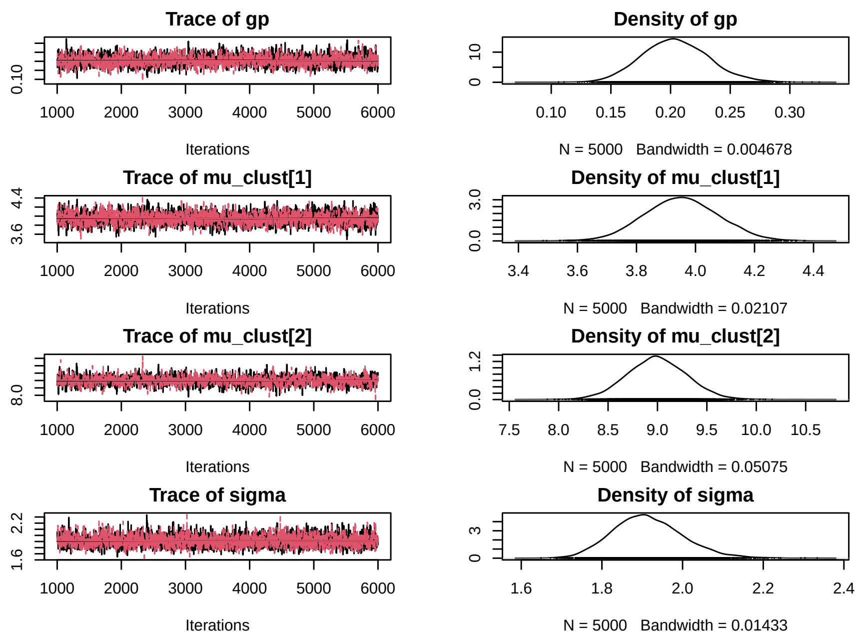
```
## mu_clust[1]    4.0  0.1    3.7    3.9    4.0    4.0    4.2    1  1900
## mu_clust[2]    9.0  0.3    8.4    8.8    9.0    9.2    9.6    1   330
## sigma          1.9  0.1    1.8    1.9    1.9    2.0    2.1    1  2000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 555.0 and DIC = 2621.9
## DIC is an estimate of expected predictive error (lower deviance is better).
```

We can plot the chain samples in order to verify that the components correctly represent the two clusters.

```r
library(coda)
jags_model <- jags.model(
  file = textConnection(model_code),
  data = model_data,
  n.chains = 2,
  n.adapt = 1000
)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 500
##    Unobserved stochastic nodes: 504
##    Total graph size: 2018
##
## Initializing model
```

```r
samps <- coda.samples(jags_model, model_params, n.iter = 5000)
par(mar = c(4, 2, 2, 4)) # correct plot size
plot(samps)
```



Trace of gp
Density of gp
Iterations
N = 5000 Bandwidth = 0.004678
Trace of mu_clust[1]
Density of mu_clust[1]
Iterations
N = 5000 Bandwidth = 0.02107
Trace of mu_clust[2]
Density of mu_clust[2]
Iterations
N = 5000 Bandwidth = 0.05075
Trace of sigma
Density of sigma
Iterations
N = 5000 Bandwidth = 0.01433

Chapter 11

Lab 1 - EDA (iris)

11.1 Iris Dataset

Download the dataset at this link (https://archive.ics.uci.edu/ml/datasets/iris) (look for the `iris.data` file) and place the file together with your R script (for simplicity). At that web page, you can also get some information regarding the origin and nature of the data.

The dataset is available as Comma-Separated Values (CSV) file, which is nothing but a plain text file where each row is a row of a table and every column is separated by a comma. To make it look more like a CSV file, rename it from `iris.data` to `iris.csv`.

11.2 Import data

Now we're ready to import the data in R and view it as a table.

> Hint: use `read_csv()` function from the `readr` library. Check the function arguments. Also, you might need to manually add the column names. To check which column names should be added and in which order, check under **"attribute information"** on the dataset reference page linked above.

```
library(tidyverse) # we will use tibble, dplyr, ggplot, readr, ...
```

```
iris_df <- read_csv("./datasets/iris.csv",
  col_names = c(
    "sepal_length", "sepal_width",
    "petal_length", "petal_width", "class"
  )
)
```

```
## Rows: 150 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): class
## dbl (4): sepal_length, sepal_width, petal_length, petal_width
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
iris_df
```

```
## # A tibble: 150 x 5
##    sepal_length sepal_width petal_length petal_width class
##           <dbl>       <dbl>        <dbl>       <dbl> <chr>
##  1          5.1         3.5          1.4         0.2 Iris-setosa
##  2          4.9         3            1.4         0.2 Iris-setosa
##  3          4.7         3.2          1.3         0.2 Iris-setosa
##  4          4.6         3.1          1.5         0.2 Iris-setosa
##  5          5           3.6          1.4         0.2 Iris-setosa
##  6          5.4         3.9          1.7         0.4 Iris-setosa
##  7          4.6         3.4          1.4         0.3 Iris-setosa
##  8          5           3.4          1.5         0.2 Iris-setosa
##  9          4.4         2.9          1.4         0.2 Iris-setosa
## 10          4.9         3.1          1.5         0.1 Iris-setosa
## # i 140 more rows
```

Answer these questions:

- How many observations does the dataset consist of?
- Which are the different classes?

```
iris_df %>%
  nrow()
```

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

```
iris_df %>%
  pull(class) %>%
  unique()
```

```
## [1] "Iris-setosa"     "Iris-versicolor" "Iris-virginica"
```

11.3 Exploratory Data Analysis (EDA)

First compute the mean and standard deviation of each measure for each class separately.

```
iris_df %>%
  group_by(class) %>%
```

```
  summarize(
    msl = mean(sepal_length), msw = mean(sepal_width),
    mpl = mean(petal_length), mpw = mean(petal_width),
    ssl = sd(sepal_length), ssw = sd(sepal_width),
    spl = sd(petal_length), spw = sd(petal_width)
  )
```

```
## # A tibble: 3 x 9
##   class             msl   msw   mpl   mpw   ssl   ssw   spl   spw
##   <chr>           <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Iris-setosa      5.01  3.42  1.46 0.244 0.352 0.381 0.174 0.107
## 2 Iris-versicolor  5.94  2.77  4.26 1.33  0.516 0.314 0.470 0.198
## 3 Iris-virginica   6.59  2.97  5.55 2.03  0.636 0.322 0.552 0.275
```

- What can you infer? Is there any measure which is more indicative of a certain class?

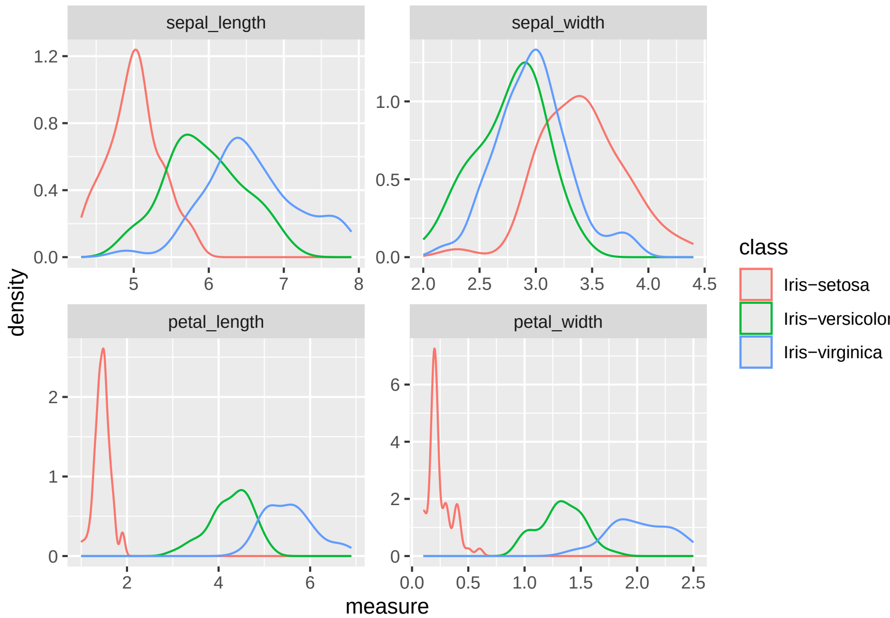
You can also plot the empirical distribution of the four measures separately, in order to better visualize how far (or close) they are from each other.

- Plot the distributions of the four measures in a 2x2 grid, differentiating the types with color encoding (optional).

  Hint: You can use `melt()` from the `reshape2` library to transform the dataset, and then plot the various densities with color encoding on the four measures.

```
library(reshape2)
iris_df %>%
  reshape2::melt(value.name = "measure") %>% # collapse all measures in one col
  # then add a column with the measure name
  ggplot() +
  geom_density(aes(measure, color = class)) +
  facet_wrap(vars(variable), nrow = 2, scales = "free")
```

```
## Using class as id variables
```

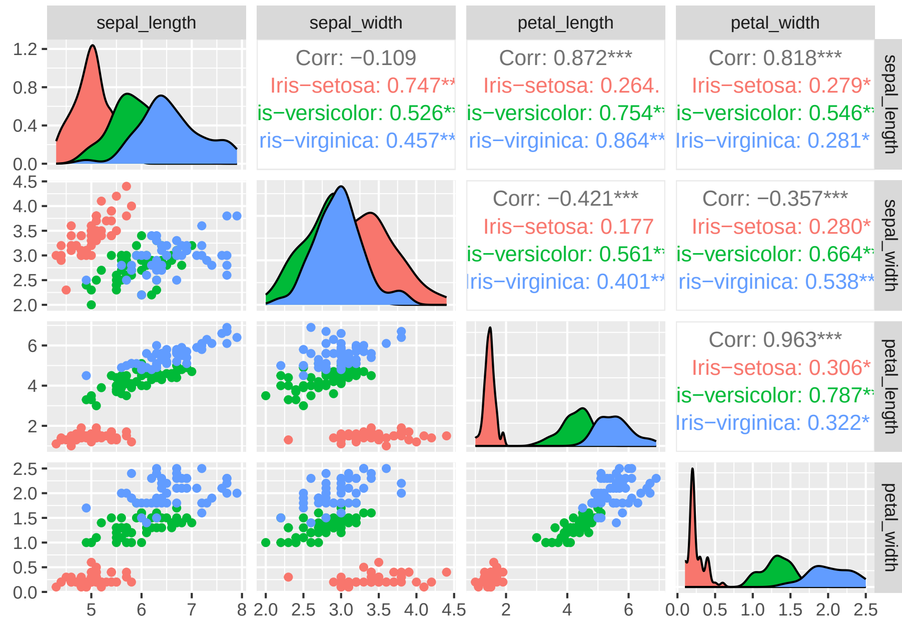


We can also visualize the four measures on a set of plots that shows the correlation between variables, two-by-two. This is easily done with a pair plot.

- Draw a pair plot (optional).

  Hint: install the package with `install.packages("GGally")` and load it with `library(GGally)`, then read the help document of the `ggpairs()` function. You can also use R base `pairs()` function.

```
library(GGally)

iris_df %>%
  ggpairs(mapping = aes(color = class), columns = 1:4)
```



11.4 Confidence intervals

Let's make our decision more statistically relevant. Select only one class, the Setosa type, and one measure, petal length.

Now build a 95% CI around the mean value of the Setosa petal length. Assume σ unknown.

Formally, we want to find x_l, x_u s.t. the true mean μ of the Setosa petal length falls in the interval with probability 95%:

$$P(x_l \leq \mu \leq x_u) = 95\% \,.$$

- Find such CI, using only R base operators (`mean`, `sd`).

To do this, remember that

- $\frac{(\bar{X}-\mu)\sqrt{n}}{s} \sim T(n-1)$ where s is the sample standard deviation
- `qt()` is the R function for the t-distribution quantile
- Validate your result by using `t.test()` to get the CI.

```
setosa_petal_length <- iris_df %>%
  filter(class == "Iris-setosa") %>%
  pull(petal_length)
```

```
alpha <- .05

nspl <- length(setosa_petal_length)
crit_t <- qt(1 - alpha / 2, nspl - 1) # critical value

# retransform the critical value multiplying the se
# and subtracting it to the sample mean
xbar <- mean(setosa_petal_length)
se <- sd(setosa_petal_length) / sqrt(nspl)
delta <- crit_t * se

ci <- c(xbar - delta, xbar + delta)
ci
```

```
## [1] 1.414689 1.513311
```

With `t.test`

```
t.test(setosa_petal_length, conf.level = 0.95)
```

```
##
##  One Sample t-test
##
## data:  setosa_petal_length
## t = 59.662, df = 49, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  1.414689 1.513311
## sample estimates:
## mean of x
##     1.464
```

Compare this confidence interval to the mean and std-dev values you got for each class for the petal length measure.

- Do you think the petal length is a good measure to differentiate between Setosa and the other two types? Why?

11.5 P-value

Imagine you get measurements in terms of all 4 indicators of 5 iris flowers belonging to the same class, but you don't know which. These are the observations.

```
x_sample <- tibble(
  sepal_length = c(4.738759, 5.545983, 5.389729, 4.549803, 5.896723),
  sepal_width = c(3.132478, 3.537232, 3.217107, 3.190097, 3.636949),
  petal_length = c(1.220472, 1.321923, 1.573662, 1.289875, 1.705737),
```

```
  petal_width = c(0.23, 0.09, 0.33, 0.22, 0.18)
)
```

- Just looking at the values of the petal length, can you take advantage of the CI you just computed in order to make a guess about the class of these flowers? (Setosa or not Setosa)

To make this guess more statistically relevant, we have to quantify our confidence.

- What is the *p-value* of this sample, only looking at the petal length, against the null hypothesis that the samples are coming from the Setosa type? We set the null hypothesis to be $H_0 : \mu = \mu_0 = \bar{x}$

  Remember that the *p-value* is defined as

$$P\left(T \leq \frac{(\bar{x} - \mu_0)\sqrt{n}}{s} | \mu = \mu_0\right)$$

- Based on the value you found, what can you say about the class of this sample?

```
# manually
mu0 <- xbar
nx <- nrow(x_sample)
t_stat <- (mean(x_sample$petal_length) - mu0) * sqrt(nx) /
  sd(x_sample$petal_length)
pt(t_stat, df = nx - 1)
```

```
## [1] 0.3380665
```

```
# with t-test
t.test(x_sample$petal_length, mu = mu0, alternative = "less")
```

```
##
##  One Sample t-test
##
## data:  x_sample$petal_length
## t = -0.44983, df = 4, p-value = 0.3381
## alternative hypothesis: true mean is less than 1.464
## 95 percent confidence interval:
##      -Inf 1.619799
## sample estimates:
## mean of x
##  1.422334
```

Now take this other sample:

```
y_sample <- tibble(
  sepal_length = c(6.303990, 6.705969, 7.795044, 7.015665, 7.056670),
```

```
  sepal_width = c(3.152411, 2.499612, 3.293934, 3.275724, 2.843923),
  petal_length = c(6.356729, 5.975576, 4.998114, 5.423811, 5.382871),
  petal_width = c(1.589342, 2.305014, 2.260900, 2.185519, 1.589370)
)
```

- What is the *p-value*?
- Is it higher or lower than 0.025?
- Could we guess the answer to this last question without computing it?

```
# manually
ny <- nrow(y_sample)
t_stat <- (mean(y_sample$petal_length) - mu0) * sqrt(ny) / sd(y_sample$petal_length)
pt(t_stat, df = ny - 1, lower.tail = FALSE)
```

```
## [1] 3.231502e-05
```

```
# with t-test
t.test(y_sample$petal_length, mu = mu0, alternative = "greater")
```

```
##
##  One Sample t-test
##
## data:  y_sample$petal_length
## t = 17.36, df = 4, p-value = 3.232e-05
## alternative hypothesis: true mean is greater than 1.464
## 95 percent confidence interval:
##  5.116134      Inf
## sample estimates:
## mean of x
##   5.62742
```

Chapter 12

Lab 2 - Linear regression (insulate)

This lab focuses on linear models and on the R functions dedicated to fitted linear models inspection.

For the following tasks, we will use a real dataset featuring two predictors and one response variable (fuel consumption). You can read some information inside the `insulate.names` text file. The dataset can be found here.

12.1 EDA (20 min)

Start by loading the `insulate.csv` data in the R environment, making sure that the type of the parsed variables is correctly inferred, then explore it using the tools seen in the lectures (e.g. dimension, summary, plots, etc.).

You might answer these questions:

- How many observations does the dataset provide?

Although it's not necessary in this case, we explicitly specify that the `when` variable needs to be parsed as factor.

```
library(dplyr)
library(readr)
# read the dataset
insulate <- read_csv("./datasets/insulate.csv", col_types = cols(
  when = col_factor(),
  temp = col_double(),
  cons = col_double()
```

```
))
insulate
```

```
## # A tibble: 56 x 3
##    when    temp  cons
##    <fct>  <dbl> <dbl>
##  1 before  -0.8   7.2
##  2 before  -0.7   6.9
##  3 before   0.4   6.4
##  4 before   2.5   6
##  5 before   2.9   5.8
##  6 before   3.2   5.8
##  7 before   3.6   5.6
##  8 before   3.9   4.7
##  9 before   4.2   5.8
## 10 before   4.3   5.2
## # i 46 more rows
```

The dataset consists of 56 records.

- What are the predictors? Are they qualitative or quantitative?

The predictors are `when, temp`. `when` is a qualitative predictor with two levels (`before, after`) while `temp` is a continuous variable in the range $[-0.8, 10.2]$.

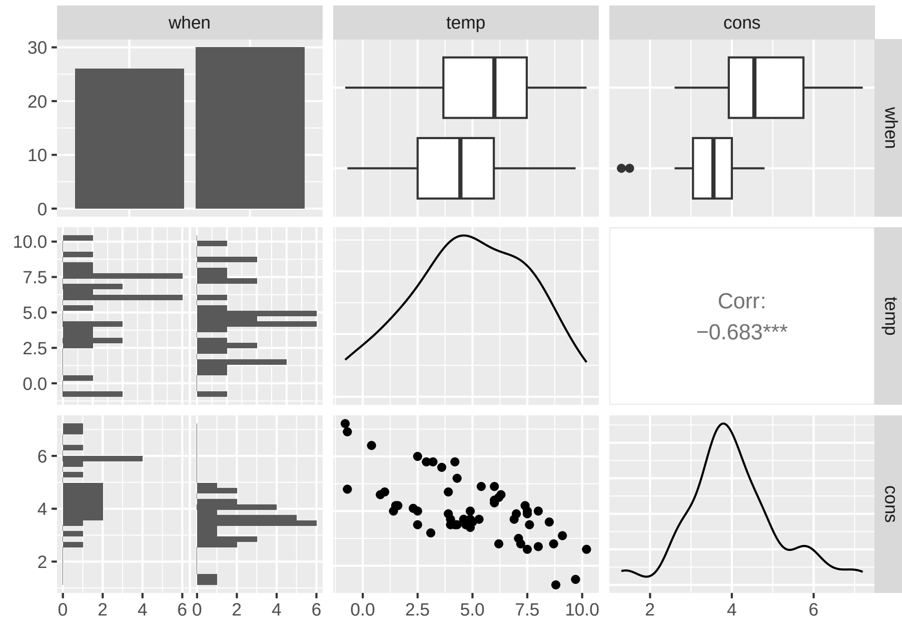
- What is the relationship between the response variable and the two predictors taken individually?
- Draw some *useful* plot and interpret what you see.

One of the most powerful plotting function is the *pairs* plot. In `ggplot` it's straightforward:

> Note: beware that when dealing with bigger datasets (more predictors), as the number of plots scales with the square of the number of columns.

```
# plot (pair plot)
library(GGally)
ggpairs(insulate)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



From these plots we can see that, as we could imagine, the consumption goes down with higher temperatures and also there is a clear difference in the consumption mean before and after insulation.

12.2 Fitting a linear model (15 min)

Now use the `lm()` function to fit two linear models: one which only uses the temperature as predictor, and one with both predictors.

- Which one is better? Why?

```
simple_lm <- lm(cons ~ temp, data = insulate)
summary(simple_lm)
```

```
##
## Call:
## lm(formula = cons ~ temp, data = insulate)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6324 -0.7119 -0.2047  0.8187  1.5327
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   5.4862     0.2357  23.275  < 2e-16 ***
## temp         -0.2902     0.0422  -6.876 6.55e-09 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8606 on 54 degrees of freedom
## Multiple R-squared:  0.4668, Adjusted R-squared:  0.457
## F-statistic: 47.28 on 1 and 54 DF,  p-value: 6.545e-09
```

```
additive_lm <- lm(cons ~ ., data = insulate)
summary(additive_lm)
```

```
##
## Call:
## lm(formula = cons ~ ., data = insulate)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.74236 -0.22291  0.04338  0.24377  0.74314
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.55133    0.11809   55.48   <2e-16 ***
## whenafter   -1.56520    0.09705  -16.13   <2e-16 ***
## temp        -0.33670    0.01776  -18.95   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3574 on 53 degrees of freedom
## Multiple R-squared:  0.9097, Adjusted R-squared:  0.9063
## F-statistic: 267.1 on 2 and 53 DF,  p-value: < 2.2e-16
```

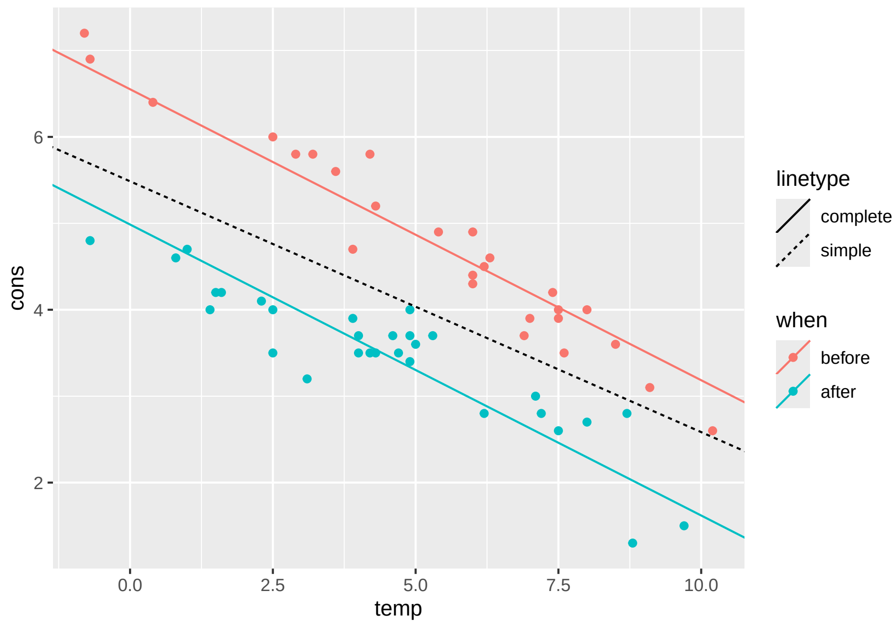
The R-squared index is much higher in the complete model, suggesting that the `when` attribute retains valuable information.

- Plot both regression lines (you can use any plotting tool that you prefer)

The additive model regression line actually consists of two (parallel) lines, one for each level of the qualitative predictor `when`. Using color encoding in ggplot, we can differentiate between the two.

```
ggplot() +
  geom_point(data = insulate, aes(temp, cons, color = when)) +
  geom_abline(mapping = aes(linetype = "simple", intercept = simple_lm$coefficients[1], slop
  geom_abline(mapping = aes(
    linetype = "complete", color = c("after", "before"),
    intercept = c(
      additive_lm$coefficients[1] + additive_lm$coefficients[2],
      additive_lm$coefficients[1]
    ),
```

```
    slope = additive_lm$coefficients[3]
  ))
```



12.2.1 Interaction (10 min)

In this setting, we are mostly interested in whether we should include the interaction between the two predictors or not. If the effects of the predictors are additive, we don't need interaction.

- Are the two effects additive? To answer this, follow the task in the next bullet point.
- Draw a plot showing the consumption trend against the temperature, separately for before and after insulation. You may (or may not) use the following code as template, which is enough to fulfill the task after replacing the ellipsis with the proper function arguments:

  Hint: remember that, in order to differentiate between classes (i.e. *before* and *after*) in the same plot, you can select the class to be interpreted as color encoding for the points/line: `aes(..., color = when)`.

```
insulate %>% # dataset
  ggplot(aes(temp, cons, color = when)) + # main ggplot call
  geom_point() + # scatter plot
  geom_smooth(method = "lm") # trend line
```

The two lines are not parallel, which suggests that an additive model is not very accurate and we might try to include interaction.

- Fit a linear model which includes interaction between `temp` and `when`.

```
interact_lm <- lm(cons ~ when * temp, data = insulate)
summary(interact_lm)
```

```
##
## Call:
## lm(formula = cons ~ when * temp, data = insulate)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.97802 -0.18011  0.03757  0.20930  0.63803
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      6.85383    0.13596  50.409  < 2e-16 ***
## whenafter       -2.12998    0.18009 -11.827 2.32e-16 ***
## temp            -0.39324    0.02249 -17.487  < 2e-16 ***
## whenafter:temp  0.11530    0.03211   3.591 0.000731 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.323 on 52 degrees of freedom
## Multiple R-squared:  0.9277, Adjusted R-squared:  0.9235
## F-statistic: 222.3 on 3 and 52 DF,  p-value: < 2.2e-16
```

- How many additional parameters does the model have to estimate?

Just one additional parameter is required.

12.3 AIC and model selection (10 min)

Looking at the R-squared index, the model with interaction seems to perform a bit better. However, this doesn't mean that the interaction model is a better choice.

- After assessing that we can compare the additive with interaction models through the ANOVA test, checking the main prerequisite (nested models), run an ANOVA test comparing the two models, analyze the result, draw a conclusion.

```
anova(additive_lm, interact_lm)
```

```
## Analysis of Variance Table
##
```

```
## Model 1: cons ~ when + temp
## Model 2: cons ~ when * temp
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1     53 6.7704
## 2     52 5.4252  1    1.3451 12.893 0.0007307 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Interaction with `*` also includes the singular variables, therefore the additive model is nested into interaction.

Another criterion used for model selection is the Akaike Information Criterion, which is defined as

$$\mathrm{AIC} = 2k - 2\ln(\hat{L})$$

where $\hat{L}$ is the maximized likelihood of the data i.e. $p(y|\hat{\beta})$. The AIC takes into consideration the model complexity (number of predictors considered), favoring small models over the bigger ones.

- Find the AIC index using the dedicated function `AIC()`. Which model is better according to the AIC? Why?

```
AIC(additive_lm, interact_lm)
```

```
##             df      AIC
## additive_lm  4 48.60465
## interact_lm  5 38.20098
```

A lower index for the interaction model means that the additional parameter is worth the higher complexity in favor of better fit (likelihood).

12.4 Prediction intervals (35 min)

Imagine we have a new unseen (*future*) row of observed measures x_f, with no true response observation (Y_f is unknown).

We are interested in making inference on the mean $x_f\beta$. As you've seen in the theory lectures, this is a particular case of the inference of $C\beta$. Being just one row, the F-type intervals can be translated to T-type intervals:

$$F_\alpha(1, n-p) = t^2_{\frac{\alpha}{2}}(n-p)$$

These are standard confidence intervals, and you can find them with `predict.lm(..., interval = "confidence")`.

- Find a prediction for the following additional rows using the best model you found so far, together with their confidence intervals.

```
when,temp
before,5.8
before,-1.0
after,4.8
after,9.8
```

With new observations, instead of making inference on the coefficients β, we can make inference on the unseen response variable Y_f. In this case we are not doing either estimation or testing, but rather just *prediction*. In *machine learning* the focus of inference is prediction: neural networks, for instance, have a large set of parameters which are learnt automatically from the data, but their values are most of the times non-interpretable, thus not useful per se, while the output of interest is $\hat{Y}_f$, eventually together with its uncertainty.

Inference on Y_f can be derived similarly to how you did with confidence intervals. Formally (but not too much in detail):

$$Y_f \sim \mathcal{N}(x_f'\beta, \sigma^2)$$

independent from $Y_1, ..., Y_n$. Under the homogeneity assumption of the future response with respect to the past,

$$x_f'\hat{\beta} = \hat{Y} \sim \mathcal{N}(x_f'\beta, \sigma^2 x_f'(X'X)^{-1}x_f)\,.$$

This leads to

$$Y_f - \hat{Y}_f \sim \mathcal{N}(0, \sigma(1 + x_f'(X'X)^{-1}x_f))\,,$$

which gives us the prediction intervals:

$$\hat{Y}_f \pm t_{\frac{\alpha}{2}}(n-p)\sqrt{\mathrm{MSR}(1 + x_f'(X'X)^{-1}x_f)}\,.$$

- Using this formula, compute the prediction intervals for *one* of the four new future observations written above (from scratch).

  Hint: We've seen how to compute the MSR (or RMS) in the first linear regression R lecture

```r
# compute RMS
e <- interact_lm$residuals
n <- nrow(insulate)
p <- length(interact_lm$coefficients)
rms <- t(e) %*% e / (n - p)

library(tibble)
# x_f with dummy variables:
```

```
# intercept whenafter   temp   whenafter:temp
# 1         0 (before)  5.8    0 (0 * 5.8)
x_f <- c(1, 0, 5.8, 0)
yhat_f <- x_f %*% interact_lm$coefficients
# note: there are several ways of getting yhat_f
# another way is to use `predict` with x_f not "dummyfied"

# get dummy x matrix
x <- with(insulate, model.matrix(~ when * temp))

# compute above formula
alpha <- .05 # level
delta <- qt(1 - (alpha / 2), df = n - p) * sqrt(rms * (1 + t(x_f) %*% solve(t(x) %*% x) %*%
lwr <- yhat_f - delta
upr <- yhat_f + delta
pred_interval <- data.frame(fit = yhat_f, lwr = lwr, upr = upr)
pred_interval
```

```
##        fit      lwr      upr
## 1 4.573043 3.912228 5.233857
```

- Check your computations using `predict.lm(..., interval = "prediction")`

```
# check that computations are correct
x_f0 <- tibble(when = "before", temp = 5.8) # plain newdata row
predict(interact_lm, newdata = x_f0, interval = "prediction")
```

```
##        fit      lwr      upr
## 1 4.573043 3.912228 5.233857
```

- Is the interval center the same as for the confidence intervals?

Yes, it's still $\hat{Y}$.

- Is the prediction interval wider or more narrow than the CI? Can you tell why (intuitively)?

```
predict(interact_lm, newdata = x_f0, interval = "confidence")
```

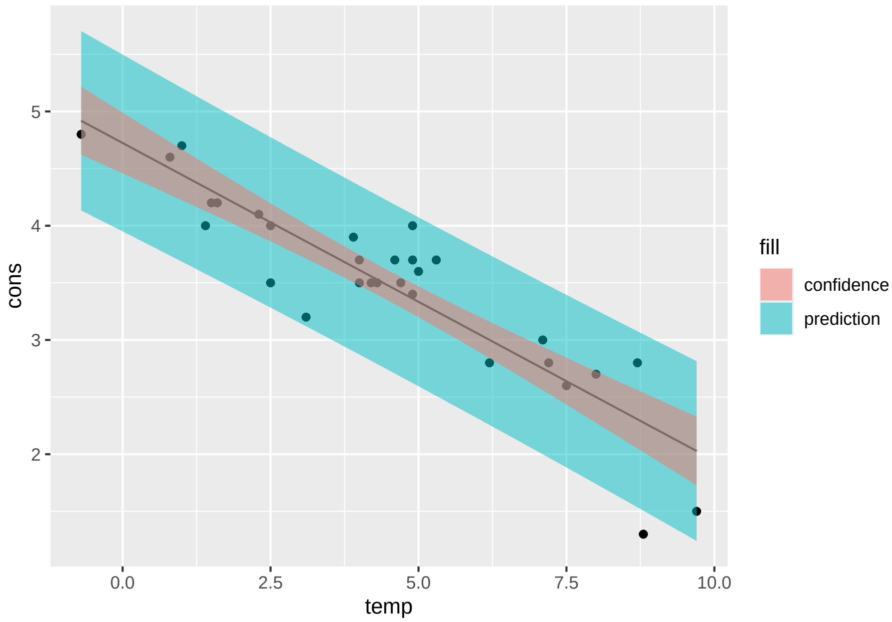
```
##        fit      lwr      upr
## 1 4.573043 4.444317 4.701768
```

It is wider because we are trying to predict where a new observation will be and that new observation has an additional standard deviation of the error term. Confidence intervals are based on only the fitted values and do not involve making a prediction. It represents the uncertainty in the "fitted" value. Prediction intervals must account for both the uncertainty in estimating the population mean, plus the random variation of the individual values. So a prediction interval is always wider than a confidence interval.

12.5 Plotting intervals

Plotting of the intervals was not required, but it might be useful to visualize them. For this purpose, we will just use the records of the `after` class.

```
# build x grid
after_insulate <-
  insulate %>%
  dplyr::filter(when == "after") %>%
  dplyr::select(-when)

# fit a simple model
after_simple_lm <- lm(cons ~ temp, data = after_insulate)

# build a vector of new data spanning the whole temp support
new_x <- tibble(temp = seq(min(after_insulate$temp), max(after_insulate$temp), by = 0.05))

# find confidence and prediction intervals for all of the range elements
# and change the column names to make them distinguishable
new_pred <- predict(after_simple_lm, newdata = new_x, interval = "prediction") %>%
  as_tibble() %>%
  rename_with(~ paste(.x, "pred", sep = "_")) #
new_conf <- predict(after_simple_lm, newdata = new_x, interval = "confidence") %>%
  as_tibble() %>%
  rename_with(~ paste(.x, "conf", sep = "_"))

# join the two interval details
new_data <- bind_cols(new_x, new_pred, new_conf)

ggplot() +
  geom_point(aes(temp, cons), data = after_insulate) + # scatter plot
  geom_line(aes(temp, fit_conf), data = new_data) + # regression line
  geom_ribbon(aes(temp, ymin = lwr_pred, ymax = upr_pred, fill = "prediction"),
    data = new_data, alpha = .5
  ) + # pred intervals
  geom_ribbon(aes(temp, ymin = lwr_conf, ymax = upr_conf, fill = "confidence"),
    data = new_data, alpha = .5
  ) # conf intervals
```



Notice how the intervals get wider when departing from the mean temperature $\bar{x}$.

Chapter 13

Lab 3 - JAGS GMM

A 5-stars hotel stocks up a whole bunch of bread loafs from a local bakery on a daily bases. Every stock of loafs counts 50 pieces, each of which varies slightly, but not trascurably, in size (volume in cm3) and weight (grams). This happens because the bakery doesn't have a precise recipe and therefore every baker produces different kinds of bread.

You get the chance of measuring weight and size of each loaf the hotel buys for an entire month and from those data you want to estimate the number of bakers that worked at the bakery in that month.

Read the data provided in the `bread.csv` file, get some insights on the nature of the data (summaries, plots, etc.) and write down a JAGS model which captures it. You should define a Gaussian Mixture Model, just like the one from previous lecture, but with the number of components K (number of bakers) as parameter.

> Hint: Maybe plot the data and get a feeling of a range of components you would like to try out. You can compare them in several ways, for instance preferring the one that gives the largest log-likelihood, printing the DIC (similar to AIC, add the flag `DIC = TRUE` in the `jags()` function call) or just by checking how many clusters give accurate estimates.

Some notes that may be useful for writing the model:

- note that `ddirich(alpha)` is the JAGS call for a Dirichlet distribution with concentration `alpha`.
- avoid the *label switching* problem by sorting the current clusters probabilities properly. The call `order(v)` is an alternative to `sort(v)`, such that `v[order(v)] == sort(v)` (gives a permutation of the indices that orders the elements of a vector in ascending order).

- to inspect the likelihood, you can define a new model node e.g. `complete_loglik` as such:

```
model {

  #
  # ... here's the main model
  #

  # likelihood
  for (i in 1:N) {
    loglikx[i] <- logdensity.norm(x[i], mux_clust[clust[i]], taux)
    logliky[i] <- logdensity.norm(y[i], muy_clust[clust[i]], tauy)
  }
  complete_loglik <- sum(loglikx) + sum(logliky)
}
```

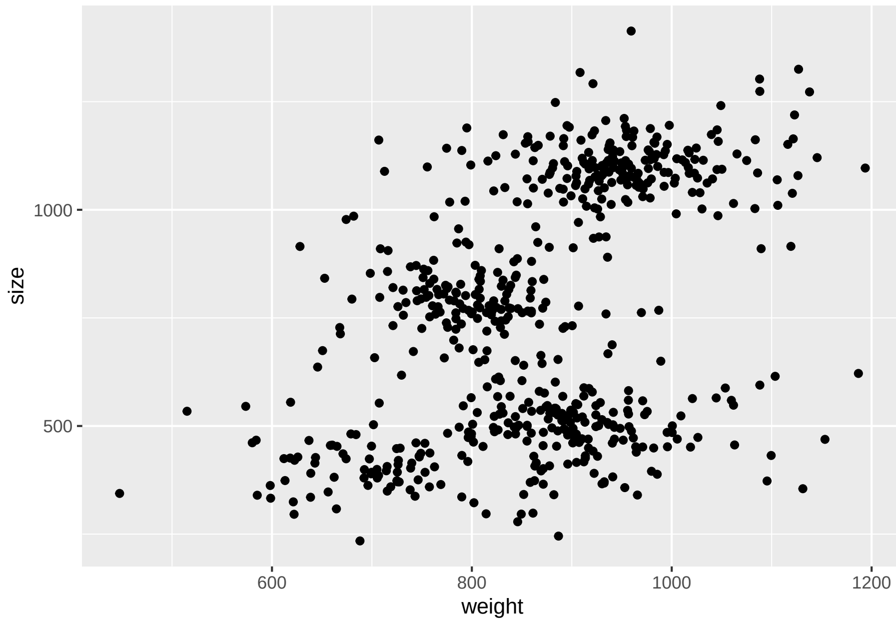
The simulation shouldn't take more than one/two minute. If it runs for too long it's either unnecessary or wrong. Decrease the number of iterations/chains.

Good luck!

```r
set.seed(42)
# read data
library(readr)
library(tibble)
library(ggplot2)

bread <- read_csv("./datasets/bread.csv")
```

```
## Rows: 600 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (2): weight, size
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
bread %>%
  ggplot(aes(weight, size)) +
  geom_point()
```




```
library(R2jags)

model_code <- "
model {
  # Likelihood:
  for(i in 1:N) {
    x[i] ~ dnorm(mux[i], taux)
    y[i] ~ dnorm(muy[i], tauy)
    mux[i] <- mux_clust[clust[i]]
    muy[i] <- muy_clust[clust[i]]
    clust[i] ~ dcat(lambda[1:K]) # categorical
  }

  # priors
  taux ~ dgamma(0.01, 0.01)
  tauy ~ dgamma(0.01, 0.01)
  sigmax <- 1 / sqrt(taux)
  sigmay <- 1 / sqrt(tauy)

  for (k in 1:K) {
    mux_clust_raw[k] ~ dnorm(0, 10^-6)
    muy_clust_raw[k] ~ dnorm(0, 10^-6)
  }

  perm <- order(mux_clust_raw) # same ordering for both!
```

```
  for (k in 1:K) {
    mux_clust[k] <- mux_clust_raw[perm[k]]
    muy_clust[k] <- muy_clust_raw[perm[k]]
  }
  lambda[1:K] ~ ddirch(ones)

  # likelihood
  for (i in 1:N) {
    loglikx[i] <- logdensity.norm(x[i], mux_clust[clust[i]], taux)
    logliky[i] <- logdensity.norm(y[i], muy_clust[clust[i]], tauy)
  }
  complete_loglik <- sum(loglikx) + sum(logliky)
}
"

K <- 4 # change this and see what's best
model_data <- list(
  x = bread$weight, y = bread$size, ones = rep(1, K),
  K = K, N = nrow(bread)
)
model_params <- c(
  "mux_clust", "muy_clust", "sigmax", "sigmay"
  # ,"clust"
  , "lambda", "complete_loglik"
)
model_inits <- function() {
  list(
    mux_clust_raw = rnorm(K, 500, 1e4), muy_clust_raw = rnorm(K, 500, 1e4),
    taux = rgamma(0.1, 0.1), tauy = rgamma(0.1, 0.1),
    clust = sample(1:K, size = nrow(bread), replace = TRUE, prob = rep(1 / K, K))
  )
}
model_run <- jags(
  data = model_data,
  parameters.to.save = model_params,
  inits = model_inits,
  model.file = textConnection(model_code),
  n.chains = 2,
  n.iter = 5000,
  n.burnin = 1000,
  n.thin = 5,
  DIC = TRUE
)
```

```
## Compiling model graph
```

```
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1200
##    Unobserved stochastic nodes: 611
##    Total graph size: 4248
##
## Initializing model
```

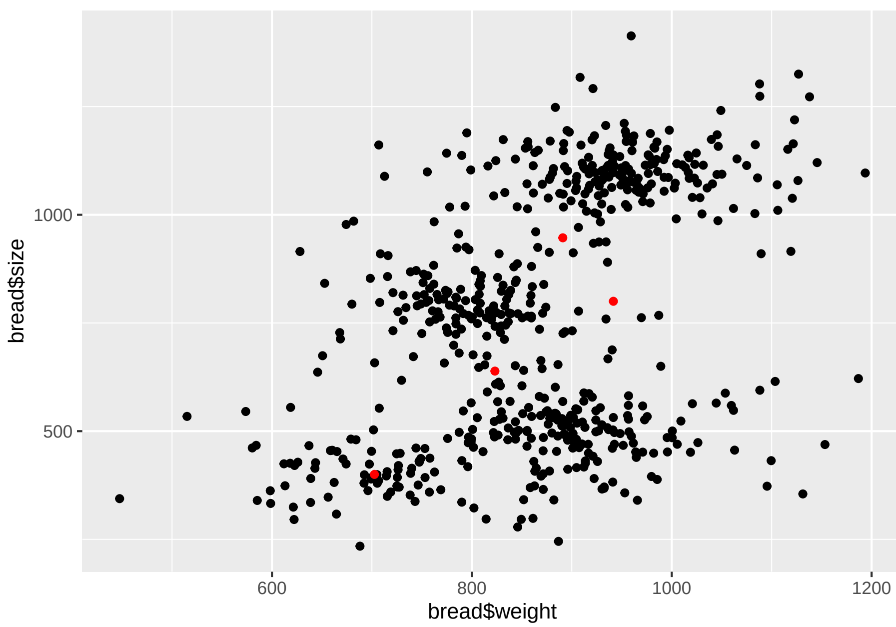
```
model_run$BUGSoutput
```

```
## Inference for Bugs model at "4", fit using jags,
##  2 chains, each with 5000 iterations (first 1000 discarded), n.thin = 5
##  n.sims = 1600 iterations saved
##                    mean    sd    2.5%     25%     50%     75%   97.5% Rhat
## complete_loglik -6940.6  49.1 -7013.7 -6987.2 -6941.2 -6892.4 -6877.1  9.1
## deviance        13881.1  98.2 13754.3 13784.8 13882.4 13974.5 14027.3  9.1
## lambda[1]           0.1   0.0     0.1     0.1     0.1     0.2     0.2  3.8
## lambda[2]           0.3   0.1     0.2     0.2     0.3     0.3     0.4  5.4
## lambda[3]           0.3   0.1     0.2     0.2     0.3     0.3     0.4  6.5
## lambda[4]           0.3   0.0     0.2     0.3     0.3     0.3     0.4  4.4
## mux_clust[1]      702.5  19.8   656.9   691.5   705.4   716.1   734.4  1.8
## mux_clust[2]      823.1  25.6   787.4   798.1   824.7   847.5   859.2  9.1
## mux_clust[3]      891.0  42.2   839.2   848.8   893.5   932.8   940.7 17.8
## mux_clust[4]      941.6   9.1   926.6   934.3   940.4   948.3   960.0  3.1
## muy_clust[1]      400.1  14.2   370.3   390.1   401.3   410.4   424.3  2.1
## muy_clust[2]      638.8 151.5   475.4   487.5   628.9   789.9   801.0 48.6
## muy_clust[3]      946.8 153.7   780.9   793.5   942.1  1100.3  1109.0 53.7
## muy_clust[4]      800.1 301.9   487.5   498.3   752.2  1101.7  1110.7 97.7
## sigmax             85.8   7.1    75.2    79.1    85.6    92.2    97.4  5.7
## sigmay             72.5   2.5    67.9    70.8    72.4    74.1    77.7  1.0
##                 n.eff
## complete_loglik     2
## deviance            2
## lambda[1]           2
## lambda[2]           2
## lambda[3]           2
## lambda[4]           2
## mux_clust[1]        4
## mux_clust[2]        2
## mux_clust[3]        2
## mux_clust[4]        3
## muy_clust[1]        4
## muy_clust[2]        2
## muy_clust[3]        2
## muy_clust[4]        2
```

```
## sigmax              2
## sigmay            120
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 330.3 and DIC = 14211.4
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```r
gmm_mcmc <- as.mcmc(model_run)
# save to pdf
# pdf(file = "./lab3/mcmc_out.pdf")
# plot(gmm_mcmc)
# dev.off()
```

```r
mux <- model_run$BUGSoutput$mean$mux_clust
muy <- model_run$BUGSoutput$mean$muy_clust

ggplot() +
  geom_point(aes(bread$weight, bread$size)) +
  geom_point(aes(mux, muy), col = "red")
```




```
## tweaking contrast
```

```
## tweaking pscl
```

Bibliography

Alan E. Gelfand, Susan E. Hills, Amy Racine-Poon, and Adrian F. M. Smith. Illustration of Bayesian Inference in Normal Data Models Using Gibbs Sampling. *Journal of the American Statistical Association*, 85 (412):972–985, December 1990. ISSN 0162-1459, 1537-274X. doi: 10.1080/01621459.1990.10474968. URL http://www.tandfonline.com/doi/abs/10.1080/01621459.1990.10474968.

Christian Kleiber and Achim Zeileis. *Applied Econometrics with R*. Springer-Verlag, New York, 2008. URL https://CRAN.R-project.org/package=AER. ISBN 978-0-387-77316-2.

David Lunn, Christopher Jackson, Nicky Best, Andrew Thomas, and D. J. Spiegelhalter. *The BUGS book: a practical introduction to Bayesian analysis*. CRC Press, Taylor & Francis Group, Boca Raton, 2013. ISBN 9781466586666. OCLC: 1053536434.