Simple data mining in R with data.table and ggplot2

Pierre Tocquin / ptocquin@uliege.be

2023

Reading CSV (or any text tabulated) data

`data.table` is a package that extends the data.frame object of base R. It comes with its own functions to read (`fread`) and write (`fwrite`) tabulated data. Those replacement functions for `read.table` and `write.table` are incredibly easy to use and time/memory efficient.

```r
# install.packages("data.table") # If the library data.table is not installed

# Loading data.table library
library(data.table)

# Reading the data and storing them in a data.table object
# stringsAsFactors = TRUE is used to transform 'chr' data into factors
# this is a good practice when 'chr' data are actually experimental modalities
d <- fread(input = "data_phototropisme_s1.csv", stringsAsFactors = TRUE)

# The npa column is also an experimental modality but because it contains numbers
# it was not converted into 'factors' by stringAsFactors. So let's do it.
d$npa <- factor(d$npa)

# Have a look at the data object
# d
```

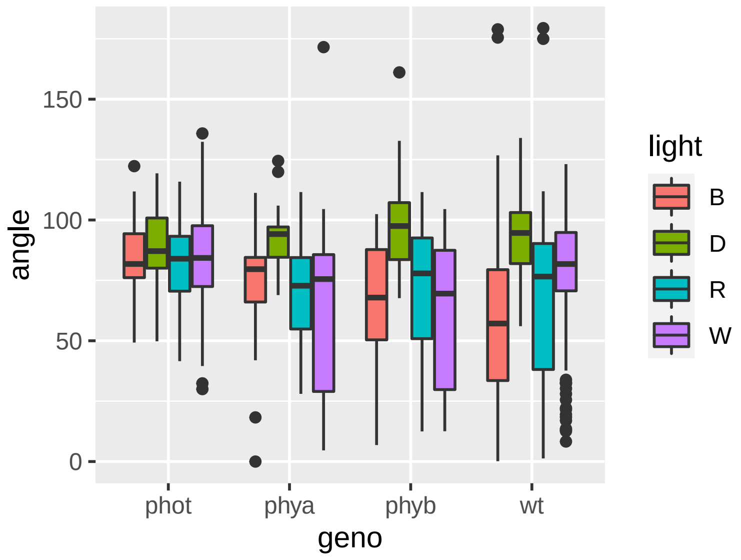
`fread` is able (most of the time) to automatically identify the type and the layout of your data (separators, headers, …). If your data are encoded with a decimal separator other than the `.`, it is necessary to set the `dec` argument of `fread`: `d <- fread(input = "data_phototropisme_s1.csv", stringsAsFactors = TRUE, dec=",")`.

Let's have a look at the data

Before going further, it is a good practice to have a global look at the dataset to check for any problems or inconsistencies. Let's use `ggplot2` for this task. Before you start coding, you should answer a few questions that will help you produce a correct graph:

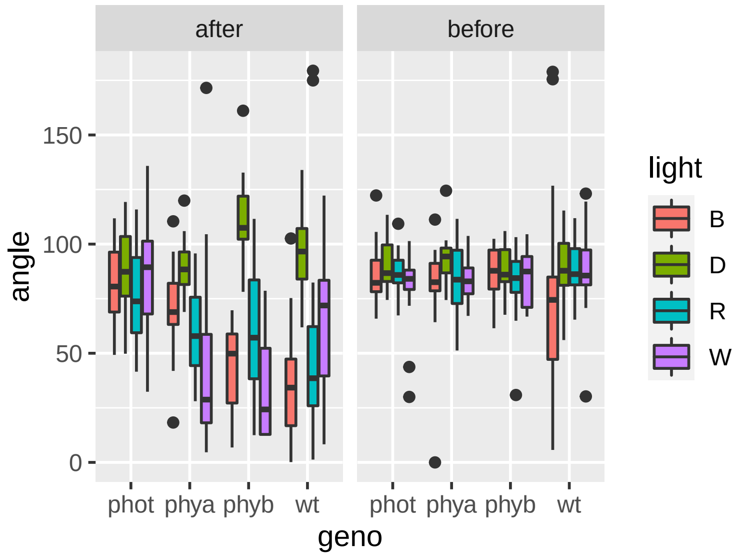
1. What data do you want to analyse ? In other words, the ones that will be plotted on the 'y' axis ? In this case, it is the `angle` column.
2. How many experimental modalities must be distinguished on the graph ? We have 5 experimental factors in total: `geno`, `light`, `repl`, `timeref` and `npa`, but on one single plot it will not be so easy to consider more than 2 factors. Let's start with the idea that we will split the `geno` data on the 'x' axis and that we will give different colors for the `light` treatments. We will then separate the `timeref`on 2 graphs. For this first overview of the data, we will not deal with `repl` (replicates A and B) or `npa`. Those data will be plotted (because we did not filter them out) but confounded in each geno~light series.

```r
# install.packages("ggplot2") # If the library is not already installed
library(ggplot2)

# the function ggplot() accepts several arguments:
# data: the data.table (or dataframe) object
# mapping: inside aes() you should at least mention x and y data (for 2D graphs)
#   depending on the type of plot other parameters can be set such as `fill`or `color`
#
# Here we plot angle (y) vs geno (x) and use 'fill' to color series depending on the light treatment
# we create a ggplot object (here 'g')
g  <- ggplot(data = d, mapping = aes(x = geno, y = angle, fill = light))
g + geom_boxplot()
```



Nice (preliminary) result but remember that, at this stage, data fro both `timeref` are confounded in each geno~light series. So let's split the results in 2 graphs, one for the data 'before' the onset of the light stimulus and another one for the data 'after' the treatment.

```r
g + geom_boxplot() + facet_grid(cols = vars(timeref))
```



This data overview helps to highlight some potential issues with the dataset that should be addressed before going further. Considering the experimental design, we expect the angle values to be around 90° for plants growing upwards (the 'normal' situation) and tend to 0° if they grow perfectly horizontally towards the lateral light source.

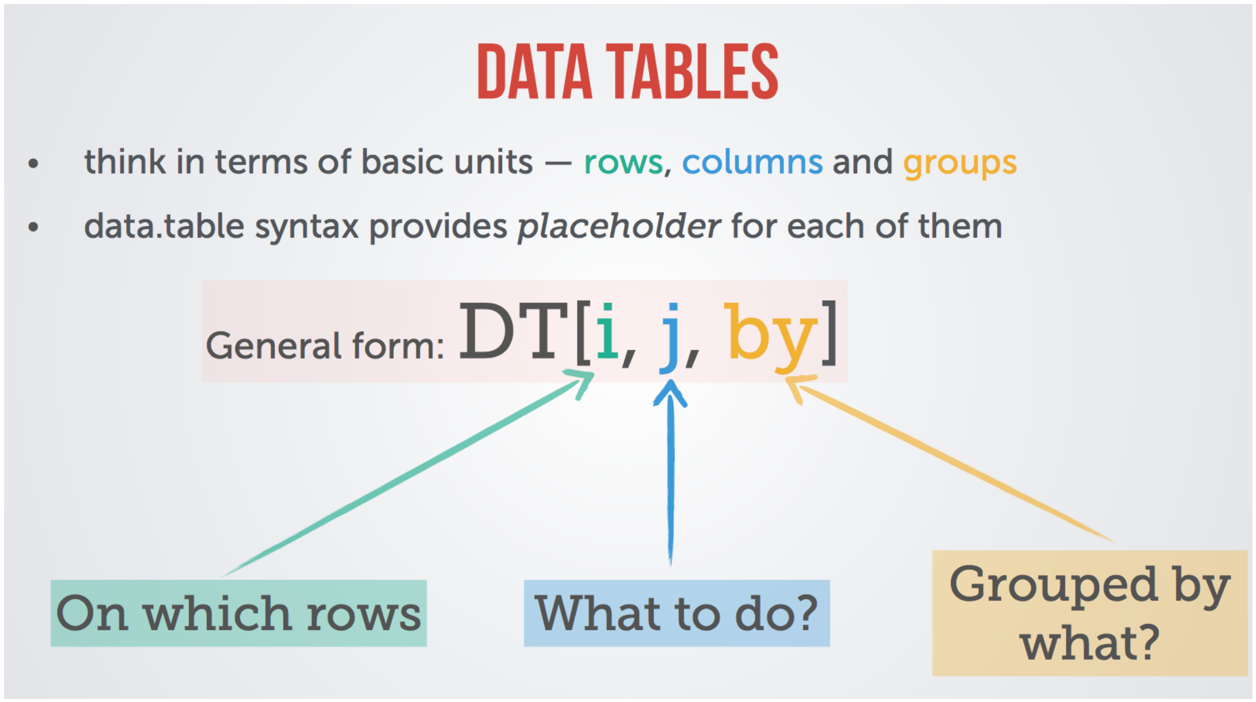
We observe on the 'before' panel that most of the series are, as expected, mostly growing upwards: the values are distributed around 90° with a limited spread, excepted for a few outliers (the black dots). The most extreme outliers (those above 150° or below 50°) should be investigated because they could be due to measurement mistakes or sample problems. If justified, those values should be excluded. It is noteworthy that the series WT in blue light is showing a large initial spread of the angle values. This is not expected because all WT series have just, at this stage, experienced the same conditions. We should check if something goes with one or the other replicates for this series.

For the same reasons, on the 'after' panel the values above 150° should be investigated and excluded if justified.

Filtering and manipulating data

`data.table` makes the filtering and manipulation of data very easy. We will first illustrate how it works by filtering the data to check for (and possibly remove) the anomalies highlighted above.

The `data.table` object comes with built-in aggregative capabilities (*via* the `by` keyword). A picture is worth a thousand words…


DATA TABLES

- think in terms of basic units — rows, columns and groups
- data.table syntax provides *placeholder* for each of them

Figure 1: The data.table general form (from https://github.com/Rdatatable/data.table/wiki)

Row filtering

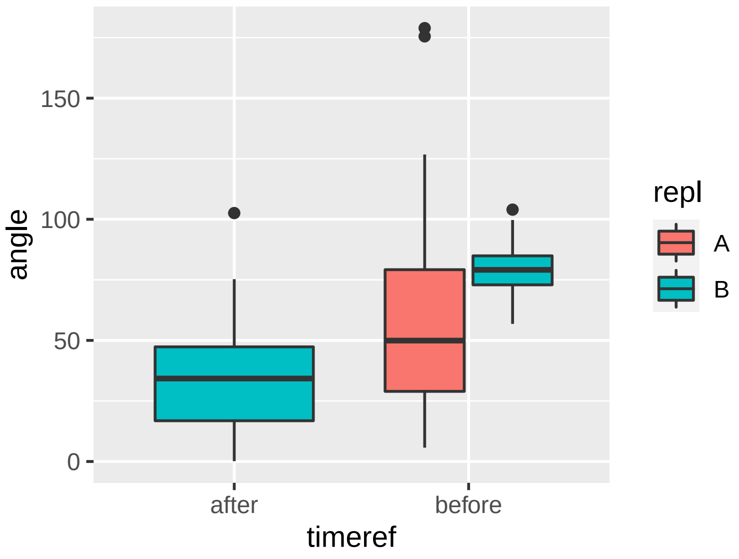
The basic *row* filtering is similar to the data.frame method, **except that column names can be used as *variables*** .

As a first example, suppose we want to filter out the angle values greater than 150°:

```r
# The data.frame version
# data[data$angle <= 150,]

# The data.table version
d[angle <= 150]

# Note that with `data.table` the ',' delimiting the *x,y* (or *i,j*) dimensions
# of the table can be omitted when *j* is empty (meaning you want to return all
# the columns without further manipulation).
```

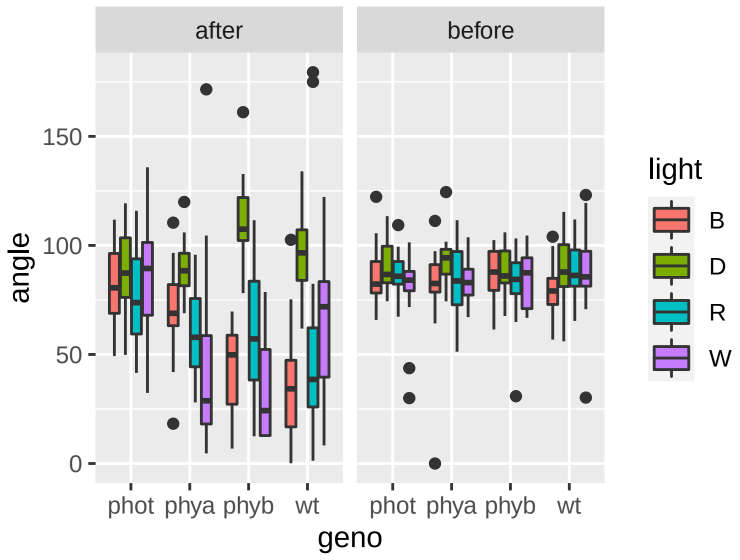
It is of course possible to make complex filtering using conditions involving multiple columns:

```r
# Filtering the rows containing data for WT exposed to blue light
# & is the logical operator AND
f1 <- d[geno == "wt" & light == "B"]

# Let's plot the result to check the data
# since we fixed geno and light, we can evaluate the 2 other factors: timeref and repl
g1 <- ggplot(data = f1, mapping = aes(x = timeref, y = angle, fill = repl))
g1 + geom_boxplot()
```



From this graph, it is clear that something went wrong with the replicate 'A' which shows an unexpected wide spread of the data, while the 'B' replicate shows the expected data for plants grown in the dark. It is safe to remove this replicate from the dataset.

```r
# First, check the conditions that select the targeted rows
# d[geno == "wt" & light == "B" & repl == "A" & timeref == "before"]
# Then, use !(...) to inverse the filtering
f2 <- d[!(geno == "wt" & light == "B" & repl == "A" & timeref == "before")]

# Make a plot to check the result
g2  <- ggplot(data = f2, mapping = aes(x = geno, y = angle, fill = light))
g2 + geom_boxplot() + facet_grid(cols = vars(timeref))
```



That's better. So, now, let's focus on the 'after' time series, and compare the replicates for each series.

```
# Starting with the filtered f2 data.table, we filter out the 'before' data
# Since we want to compare the replicates, we will make one graph by genotype
# and use the light treatement as the 'x' axis and the replicate as the color
f3 <- f2[timeref == "after"]

g3 <- ggplot(data = f3, mapping = aes(x = light, y = angle, fill = repl))
# g3 + geom_boxplot() + facet_grid(cols = vars(geno))
```

We could highlight a few conditions were the replicates show differences but nothing clear enough to filter out any series without a deeper analysis of the raw data (and probably the pictures themselves). However, if the objective is to produce bar plots with means and standard deviations, we should be careful with outliers such as those we observe for the WT & red light series. Two values close to 180° are clearly weird since the other values for this series are spread around 40°. So, we should filter them out.

```
# First check the conditions that select the outliers
# f2[timeref == "after" & geno == "wt" & light == "R" & angle > 150]
# Then, filter them out
filtered_data <- f2[!(timeref == "after" & geno == "wt" & light == "R" & angle > 150)]

# Check the result
g4 <- ggplot(data = filtered_data[timeref == "after"],
             mapping = aes(x = light, y = angle, fill = repl))
# g4 + geom_boxplot() + facet_grid(cols = vars(geno))
```

Better graphs by ordering the factors

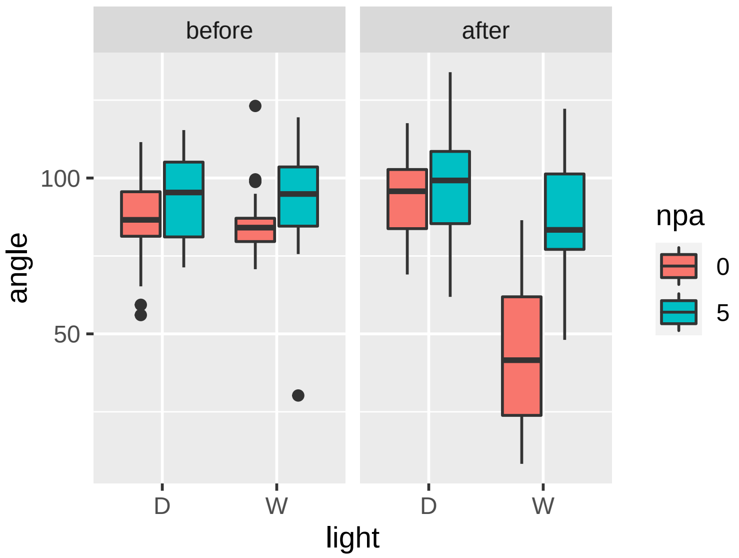
On the previous plots, you may have noticed that the series are plotted in alphabetical order: from left to right, the genotypes are phot, phya, phyb and wt and the light are B, D, R and W. This is not the best way to present the result because you should always organise the graphs from the left to the right, starting with the control conditions, because they are the obvious starting point of any comparison we want to make.

```
# You re-order the factor by using the factor function and giving to its levels argument
# a vector of values in the order you want (make sure to use the actual spelling of the factors)
filtered_data$geno    <- factor(x = filtered_data$geno,
                                levels = c("wt","phot","phya","phyb"))
filtered_data$light   <- factor(x = filtered_data$light,
                                levels = c("D","W","B","R"))
filtered_data$timeref <- factor(x = filtered_data$timeref,
                                     levels = c("before", "after"))

# check the result (compare with g4)
g5 <- ggplot(data = filtered_data[timeref == "after"], mapping = aes(x = light, y = angle, fill = repl))
# g5 + geom_boxplot() + facet_grid(cols = vars(geno))
```

Data manipulation

Now that we have cleaned and ordered our data, let's focus on one analysis: the impact of npa (an inhibitor of auxin transport) on the phototropism. So, first, we select the corresponding data, i.e. WT in dark or lateral white light, with or without npa and at both time points.

```
npa_data <- filtered_data[geno == "wt" & light %in% c("W", "D")]

# A quick glance at the results
g6 <- ggplot(data = npa_data, mapping = aes(x = light, y = angle, fill = npa))
g6 + geom_boxplot() + facet_grid(cols = vars(timeref))
```



Looking at the 'after' panel, the result is quite clear and using boxplot seems perfectly suited, but for the purpose of learning data manipulation with data.table we will calculate means and standard deviations and plot the resulting data with a bar plot.

The *j* placeholder of the data.table object make possible direct manipulation of data by using column names as variables.

```
# Note the ',' indicating that the first position 'i' is empty, i.e. no filtering.
npa_data[, mean(angle)]
```

The *j* placeholder accepts a **list** of arguments, making possible multiple and simultaneous data manipulations.

```
# npa_data[, list(mean(angle), sd(angle))]
# The same with direct renaming of the new columns
npa_data[, list(mean=mean(angle), sd=sd(angle))]
```

This is not what we want, since we need to calculate mean and standard deviation for each experimental series (light * npa). One option would be to use the *i* placeholder to filter the data and make multiple independent (manual) calculations:

```
# Bad example !!!
npa_data[light == "D" & npa == 0, list(mean=mean(angle), sd=sd(angle))]
npa_data[light == "D" & npa == 5, list(mean=mean(angle), sd=sd(angle))]
# ...
```

That's not the good way (neither straightforward) of doing it. You should rather us the third placeholder of the data.table object, *by*, to aggregate data before the calculation at the *j* placeholder is done. *by* accepts as an argument a **list** of one or more column names.

```
npa_data[, list(n=length(angle), mean=mean(angle), sd=sd(angle)), by=list(npa, light)]
```

```
##    npa light  n     mean       sd
## 1:   0     W 68 63.73974 27.31103
## 2:   5     W 60 90.49246 16.56949
## 3:   0     D 60 90.43083 13.72857
## 4:   5     D 58 95.42534 17.06773
```

'*On place*' update of `data.table` objects

The `:=` operator can be used at the *j* placeholder to add, remove and update columns by reference. For example, if you want to transform, in place, the angle values to make 0° the upward direction and -90° the right direction, you should do:

```
npa_data[, angle := angle-90]
```

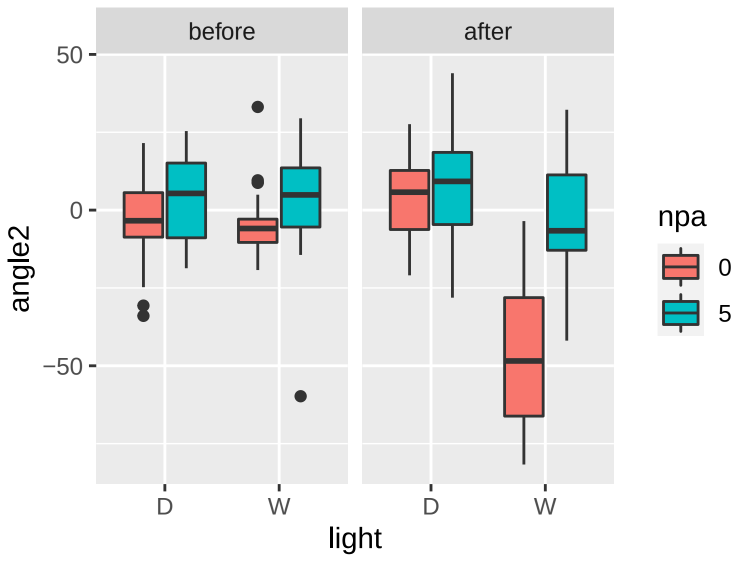
Note that, by using the `:=` operator, you modify your original `data.table` object... even if you do not use explicitely the `<-` assignement operator, like in `npa_data <- npa_data[, angle := angle-90]`.

This operator can also be used to add a new column or remove an existing column:

```
# re-create the npa_data object since the previous command modified it
npa_data <- filtered_data[geno == "wt" & light %in% c("W", "D")]
# add a column with radian values
npa_data[, angle2 := angle-90]
# optionnaly remove the angle column
# data[, angle := NULL]
```

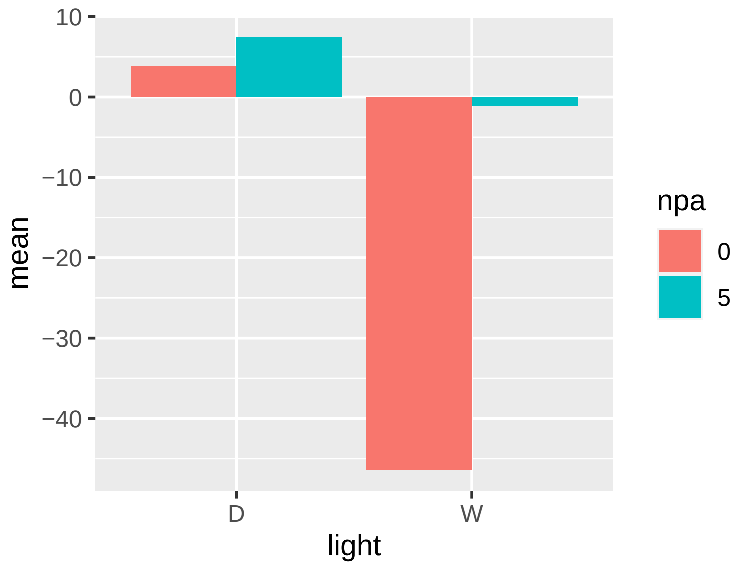
or in one operation:

```
# re-create the npa_data object since the previous command modified it
npa_data <- filtered_data[geno == "wt" & light %in% c("W", "D")]

npa_data[, c("angle2", "angle") := list(angle-90, NULL)]
# equivalent to
# npa_data[, `:=` (angle = angle-90, angle = NULL)])

# A quick glance at the results
g7 <- ggplot(data = npa_data, mapping = aes(x = light, y = angle2, fill = npa))
g7 + geom_boxplot() + facet_grid(cols = vars(timeref))
```



Let's use this new `angle2` data to calculate means and standard deviations and produce a bar plot.

```
# We limit the plot to the 'after' time series
summary <- npa_data[timeref == "after",
                    list(n=length(angle2), mean=mean(angle2), sd=sd(angle2)),
                    by=list(npa, light)]

g8 <- ggplot(data = summary, mapping = aes(x = light, y = mean, fill = npa))
# g8 + geom_col()

# By default, multiple bars occupying the same x position will be stacked atop one another by position_
# If you want them to be dodged side-to-side, use position_dodge() or position_dodge2().
# Finally, position_fill() shows relative proportions at each x by stacking the bars and
# then standardising each bar to have the same height.

g8 + geom_col(position= position_dodge())
```



Now, let's improve the plot by adding/updating key elements

1. Add error bars

```
# Note the use of position_dodge() also with geom_errorbar to correctly position the error bars
# g8 + geom_col(position= position_dodge()) +
#   geom_errorbar(mapping = aes(ymin = mean-sd, ymax=mean+sd), position = position_dodge())

# Controlling the width of the error bars and the centering on the bars
# g8 + geom_col(position= position_dodge()) +
#   geom_errorbar(mapping = aes(ymin = mean-sd, ymax=mean+sd, width = 0.25),
position = position_dodge(0.9))
# It would be better to plot only one side of the error bars, but it required to precompute ymax
# So let's update the summary table
summary[mean < 0, ymax := mean-sd]
summary[mean >= 0, ymax := mean+sd]
# or in one operation
summary[, ymax := ifelse(mean < 0, mean-sd, mean+sd)]

# use it to plot the well oriented half of the error bar
g8 + geom_col(position= position_dodge(), colour = "black") +

  geom_errorbar(mapping = aes(ymin = mean, ymax=ymax, width = 0.25),
                position = position_dodge(0.9))
# but remember that, in my opinion, the boxplot is much more suited here.
```

2. Update axis titles

```
# Note the use of \n to make a new line
g8 + geom_col(position= position_dodge(), colour = "black") +
  geom_errorbar(mapping = aes(ymin = mean, ymax=ymax, width = 0.25), position = position_dodge(0.9)) +
  xlab("Light condition") +
  ylab("Hypocotyl orientation (°)\n relative to upward direction (0°)")
```

2. Update axis and tick labels

```
# Note the use of \n to make a new line
g8 + geom_col(position= position_dodge(), colour = "black") +
  geom_errorbar(mapping = aes(ymin = mean, ymax=ymax, width = 0.25), position = position_dodge(0.9)) +
  xlab("Light condition") +
  ylab("Hypocotyl orientation (°)\n relative to upward direction (0°)") +
  scale_y_continuous(breaks = c(30, 0, -30, -60, -90)) +
  scale_x_discrete(labels = c("Dark", "Lateral white light"))
```

3. Improve the legend

```
# Note the use of \n to make a new line
g8 + geom_col(position= position_dodge(), colour = "black") +
  geom_errorbar(mapping = aes(ymin = mean, ymax=ymax, width = 0.25), position = position_dodge(0.9)) +
  xlab("Light condition") +
  ylab("Hypocotyl orientation (°)\n relative to upward direction (0°)") +
  scale_y_continuous(breaks = c(30, 0, -30, -60, -90)) +
  scale_x_discrete(labels = c("Dark", "Lateral white light")) +
  scale_fill_manual(labels = c("0 µM", "5 µM"), values = c("white", "red"), name = "NPA conc.")
```