An (opinionated) introduction to R for biologists

Analysing qPCR data using R

Carlos Ronchi

2022-08-01

Table of contents

Welcome **4**

I First steps **5**

1 Prerequisites **6**

2 Loading your data **7**
2.1 Formats, formats and more formats... 7
2.2 How to load your data 7
2.3 Checking the data 8
2.4 Checkpoint 9

3 Cleaning your data **11**
3.1 Loading libraries and files 11
3.2 Formatting your column names 11
3.3 Changing values in a column 12
3.4 Checkpoint 13

II Exploring **14**

4 Preparing the data **15**
4.1 Loading libraries and files 15
4.2 Exploring some statistics 15
4.3 Summarizing the data 17
4.4 Checkpoint 18

5 Plotting **19**
5.1 Visualizing cycle thresholds 19
5.2 Are there any outliers? 21

III Modeling **22**

6 Normalizing the data **23**
6.1 delta CT 23
6.2 Exploring normalized data 24
6.3 Checkpoint 26

7 Answering questions **27**
7.1 First question 27
7.2 Second question 29

8 Conclusion **31**

IV Wrap up **32**

9 Generating a report **33**

Welcome

Welcome to An (opinionated) introduction to R for biologists! In this short book we will show you how to use the basics of R and the tidyverse to analyse your qPCR. We will go over data manipulation, plotting and statistical analysis, all in a context of qPCR data.

Since this book is just a brief introduction with a focus on R, to learn more in general check the book Hands-On Programming with R. Now, if you are familiar with R already, but want to learn new tricks and usefull tools, check the book R for Data Science. The authors go over the data science tasks, some of the mentioned in this book, while going deeper into the functions mentioned here.

If you spot any errors or have any questions, feel free to raise an issue on the github repository: issues.

Part I

First steps

1 Prerequisites

2 Loading your data

The first step in any data analysis is to open up your favorite software and load the data up. You are probably familiar with user interfaces, where you click some buttons, you upload your excel file and *voilá*, data is ready to analyse.

In this chapter I will teach you how to format your qPCR data so you can load it easily to R. I will also show you how to load the data itself.

> ⚠ Warning
>
> If you are using RStudio, you can also click some buttons and load your data, without writing any code. I strongly recommend you to **not** do this. In the next sections I will show how you should be doing.

2.1 Formats, formats and more formats...

If you are here, you are probably used to excel and the format xlsx. R can open these files, however, it is easier to run your analysis if you have your data in the *csv* format. For qPCR specially, each qPCR machine, the excel sheet will be different and not in a standard format. Usually it is a table with several cells containing information regarding your run and other parameters.

Given your fresh table from the qPCR machine, we will extract some columns. In qPCR experiments, usually there are three technical replicates for your samples. This is reflected in how the data is saved.

An example table is shown in the data folder.

2.2 How to load your data

If you saved your table as a csv file, then we have several options to load the file. You can use the function `read.csv` or `read.table` from base R to open the tables. The way to use these functions is shown below.

```
qpcr <- read.csv("../data/qPCR.csv")
qpcr <- read.table("../data/qPCR.csv", sep = ",", header = TRUE)
```

Note that for `read.table` we need to specify the separator, in this case it is a comma (",") and we also specify `HEADER = TRUE`. This means the first row of your table is the header of your data frame. In general if you have a csv file, it is easier to run `read.csv`.

2.3 Checking the data

When loading the data, it is very important to check if it was imported successfully or any problem happened. A package that we will be using throughout this book is `dplyr`. Within `dplyr` we have some functions that help us deal data frames in a very intuitive way. The first function we will use here to check the data is `glimpse`. We load the package here, but it is best practice to load all the packages you will be using at the beginning of your analysis.

```
# first we start by loading the library
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
```

```
# we now use the function glimpse on our dataframe
glimpse(qpcr)
```

```
Rows: 72
Columns: 4
$ sample.ID <int> 20, 20, 20, 21, 21, 21, 22, 22, 22, 23, 23, 23, 24, 24, 24, ~
$ group     <chr> "WT C", "WT C", "WT C", "WT C", "WT C", "WT C", "WT C", "WT ~
$ gene      <chr> "AKT", "AKT", "AKT", "AKT", "AKT", "AKT", "AKT", "AKT", "AKT~
$ ct        <dbl> 16.49, 16.44, 16.65, 16.50, 16.74, 16.68, 16.96, 16.92, 17.0~
```

The function `glimpse` displays a summary of each column of your dataframe and their data types. We see that `sample_id` is an integer, `group` and `gene` are characters and finally the `ct` values are doubles, meaning they are numeric. So far the data looks good.

Another way to use the function `glimpse` and other functions from the `dplyr` package is using the pipe `%>%`. The way to interpret the pipe is the following. Given a dataframe, we pipe the dataframe to the next function. The syntax is shown in the code block below.

```
qpcr %>% glimpse
```

```
Rows: 72
Columns: 4
$ sample.ID <int> 20, 20, 20, 21, 21, 21, 22, 22, 22, 23, 23, 23, 24, 24, 24, ~
$ group     <chr> "WT C", "WT C", "WT C", "WT C", "WT C", "WT C", "WT C", "WT ~
$ gene      <chr> "AKT", "AKT", "AKT", "AKT", "AKT", "AKT", "AKT", "AKT", "AKT~
$ ct        <dbl> 16.49, 16.44, 16.65, 16.50, 16.74, 16.68, 16.96, 16.92, 17.0~
```

The output is the same as before, the difference is that we can use other functions and chain them together. For example, if we want to see only the first 5 rows of the table and then select the row with maximum ct value, we can use the function `head` followed by `slice_max`.

```
qpcr %>%
    head %>%
    slice_max(order_by = ct) # specify the column by passing the name without quotes
```

```
  sample.ID group gene    ct
1        21  WT C  AKT 16.74
```

There is a flow of code when chaining functions like this. By also writing code like this, it is easier to modify or add new steps to the chain.

2.4 Checkpoint

We now use `readRDS` to save our current matrix to load it up in the next chapter. The checkpoints are not necessary. As this is a simple analysis, you can run everything in a single script and loaded in your memory. The good thing about checkpoints is that you can save heavy calculations so you do not need to perform them again.

```
saveRDS(qpcr, "../checkpoints/loading/qpcr.rds")
```

> **Tip**
>
> `readRDS` is a function that lets you save R objects into your computer. It is extremely handy when you want to save expensive calculations or continue your analysis later. Note you are only saving one object.

> **Tip**
>
> If you are using quarto markdown or Rmarkdown, there is a chunk option that you can use to not rerun that chunk, namely the `cache` option.

For each checkpoint, we save them in a specific folder for each chapter inside `checkpoints`, in the root folder. This ensures we know where the data was generated by pointing to the name of the chapter.

3 Cleaning your data

In this chapter we will go over how you can clean your data before you start any analysis. Doing this before anything else will save you a lot of time. It is very frequent that you need to change a variable name or create a new column in your dataframe. By performing the cleaning and modifications before doing any plotting, it helps to ensure you will be using clean data and it will make your life easier.

3.1 Loading libraries and files

As discussed before, we start by loading the libraries we will use. In this section we will go over the packages `janitor`, `stringr` and `tidyr`.

```
library(dplyr)
library(stringr)
library(janitor)
library(tidyr)
```

And we load up our qPCR data saved from the checkpoint. To do this we use the function `readRDS`. For this we only specify the path to where the checkpoint was saved.

```
qpcr <- readRDS("../checkpoints/loading/qpcr.rds")
```

3.2 Formatting your column names

Whenever doing analysis on R it is important to have sound column names. They make data wrangling easier. Moreover, when you repeat the analysis over and over again, having sane names for your columns makes your code cleaner. There is a very neat package in R that helps cleaning your column names and suggesting new names: `janitor`. In this package there is the function `clean_names` that changes all column names to lower case, change spaces and dots to underscores and much more. The idea is that you have very clear names without any non conventional character.

The qPCR dataframe has the following columns:

```
colnames(qpcr)
```

```
[1] "sample.ID" "group"     "gene"      "ct"
```

The column `sample.ID` has both lower and upper case. This does not help when referencing and makes reading more difficult.

```
qpcr <- qpcr %>% janitor::clean_names()

colnames(qpcr)
```

```
[1] "sample_id" "group"     "gene"      "ct"
```

The column was changed from `sample.ID` to `sample_id`.

3.3 Changing values in a column

We saw previously that the column group contains the knockout and treatment assignments. This would be better if it was in different columns. Notice that these two information are separated by a space in the `group` column. To split them, we use the function `separate` from `tidyr`.

```
qpcr <- qpcr %>%
    # set the argument remove to FALSE so the column is not removed
    # from the dataframe
    separate(col = "group", into = c("genotype", "treatment"), remove = FALSE)

qpcr %>% head
```

```
  sample_id group genotype treatment gene    ct
1        20  WT C       WT         C  AKT 16.49
2        20  WT C       WT         C  AKT 16.44
3        20  WT C       WT         C  AKT 16.65
4        21  WT C       WT         C  AKT 16.50
5        21  WT C       WT         C  AKT 16.74
6        21  WT C       WT         C  AKT 16.68
```

Note that the samples have an ID, and they are represented by integers. Actually it would be better if this columns is represented by factors. This is actually very important when doing linear modelling. If a column is numeric, or integers, R will estimate the parameters as if the variable is continuous, whereas it should be a group for example.

To modify one column we can use the function `mutate` from `dplyr`. We use the function `factor` to convert a numeric vector into a categorical vector.

```
qpcr <- qpcr %>%
    mutate(sample_id = factor(sample_id))

class(qpcr[, "sample_id"])
```

```
[1] "factor"
```

3.4 Checkpoint

We are ready to proceed to the next steps. Before that we save once again our data. You can either overwrite or create a new object. For the sake of clarity, we always save a new object. Sometimes this is not useful because the objects are big, so you want to actually overwrite the previous rds object.

```
saveRDS(qpcr, "../checkpoints/cleaning/qpcr.rds")
```

Part II

Exploring

4 Preparing the data

> Describe data needs to be in special format to do stuff in R, explain pivoting. Explain groupping as well, which makes it easier to do average calculations

In this chapter we start checking and preparing the data for future exploration. The qPCR data is special, as we always have technical replicates, meaning we have multiple measures of the same sample material. One common step is to average the cycle threshold values of each sample to get a final value. This usually works pretty well, provided there is no outlier. Therefore, it is important to explore the data and understand what it looks like.

The following sections will show you how to do some preliminary checks of the data and how to create summaries so we can do some plots and statistical analysis.

4.1 Loading libraries and files

```
library(dplyr)
library(tidyr)
library(gtsummary)

qpcr <- readRDS("../checkpoints/cleaning/qpcr.rds")
```

4.2 Exploring some statistics

We first start by checking the structure of the dataset, number of samples by ID, treatment, genotype and more. For this we can pipe the `qpcr` dataframe to the function `tbl_summary` from the package `gtsummary`. A nice package to use when writing reports.

```
qpcr %>% gtsummary::tbl_summary()
```

```
Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.
```

Characteristic	**N = 72**
sample_id	
20	6 (8.3%)
21	6 (8.3%)
22	6 (8.3%)
23	6 (8.3%)
24	6 (8.3%)
25	6 (8.3%)
26	6 (8.3%)
27	6 (8.3%)
28	6 (8.3%)
29	6 (8.3%)
30	6 (8.3%)
31	6 (8.3%)
group	
KO C	18 (25%)
KO T	18 (25%)
WT C	18 (25%)
WT T	18 (25%)
genotype	
KO	36 (50%)
WT	36 (50%)
treatment	
C	36 (50%)
T	36 (50%)
gene	
AKT	36 (50%)
GAPDH	36 (50%)
ct	17.28 (17.15, 17.65)

The next step is to group the data to understand the number of technical replicates per sample. Sometimes, due to the way the experiment runs, some technical replicates are lost. We just want to make sure what is available for the analysis.

Two very important functions are used here to summarise the information. We first need to group the rows that we want to summarise. In this case, we want to group by gene, sample ID and also the treatment + genotype. The `group_by` is the one to be used. Usually after using `group_by`, one can summarise the data using the function `summarise`. In there you create columns similarly to the function `mutate`, but applying functions to the columns of the grouped dataset. In this case, we use the function `n()` to count the number of rows in the grouped dataframe.

n	n_n	percent
3	24	1

```
nb_technical_replicates <- qpcr %>%
    group_by(group, gene, sample_id) %>%
    summarise(n = n())
```

```
`summarise()` has grouped output by 'group', 'gene'. You can override using the
`.groups` argument.
```

The function `tabyl` from the package `janitor` counts the frequency of the specified column. In this case, we want to know the frequency of the number of technical replicates.

```
nb_technical_replicates %>%
    janitor::tabyl(n) %>%
    kableExtra::kbl() %>%
    kableExtra::kable_classic(full_width = FALSE)
```

We see that all samples have 3 technical replicates. If you want to inspect the table, the function `head` shows the first 5 rows of the table.

```
head(nb_technical_replicates)
```

```
# A tibble: 6 x 4
# Groups:   group, gene [2]
  group gene  sample_id     n
  <chr> <chr> <fct>     <int>
1 KO C  AKT   26            3
2 KO C  AKT   27            3
3 KO C  AKT   28            3
4 KO C  GAPDH 26            3
5 KO C  GAPDH 27            3
6 KO C  GAPDH 28            3
```

4.3 Summarizing the data

Now that we saw the data looks fine at the first level, the average CT values can be calculated. To do this we again use the function `group_by` and `summarise` from the `dplyr` package.

```r
summarised_qpcr <- qpcr %>%
    group_by(sample_id, gene, group, genotype, treatment) %>%
    summarise(
        mean = mean(ct),
        sd = sd(ct)
    )
```

```
`summarise()` has grouped output by 'sample_id', 'gene', 'group', 'genotype'.
You can override using the `.groups` argument.
```

4.4 Checkpoint

```r
saveRDS(summarised_qpcr, "../checkpoints/preparing/summarised_qpcr.rds")
```

5 Plotting

In this section we will show how to explore qPCR data through plotting. Whenever doing data analysis, it is very important to explore data through data summaries and plots. This way you learn and get a better feeling for the generated data.

If you have been following the previous chapters, the data will be ready to do the plotting and data exploration. Here we will introduce the library `ggplot2` and we will combine with the pivoting functions we previously learned.

```
library(dplyr)

library(ggplot2)
```

```
qpcr <- readRDS("../checkpoints/cleaning/qpcr.rds")
summarised_qpcr <- readRDS("../checkpoints/preparing/summarised_qpcr.rds")
```

5.1 Visualizing cycle thresholds

The first step is to understand the cycle thresholds obtained in the data, even before normalizing them. This is important as we might find outliers and we check how good the housekeeping genes are.

Moreover, one important step of the analysis is to check the absolute values of the cycle thresholds. Sometimes the fold changes can be misleading if there are high values of cycle thresholds.

Since qPCR data usually is already in the long format[1], we can directly apply the whole `ggplot2` machinery.

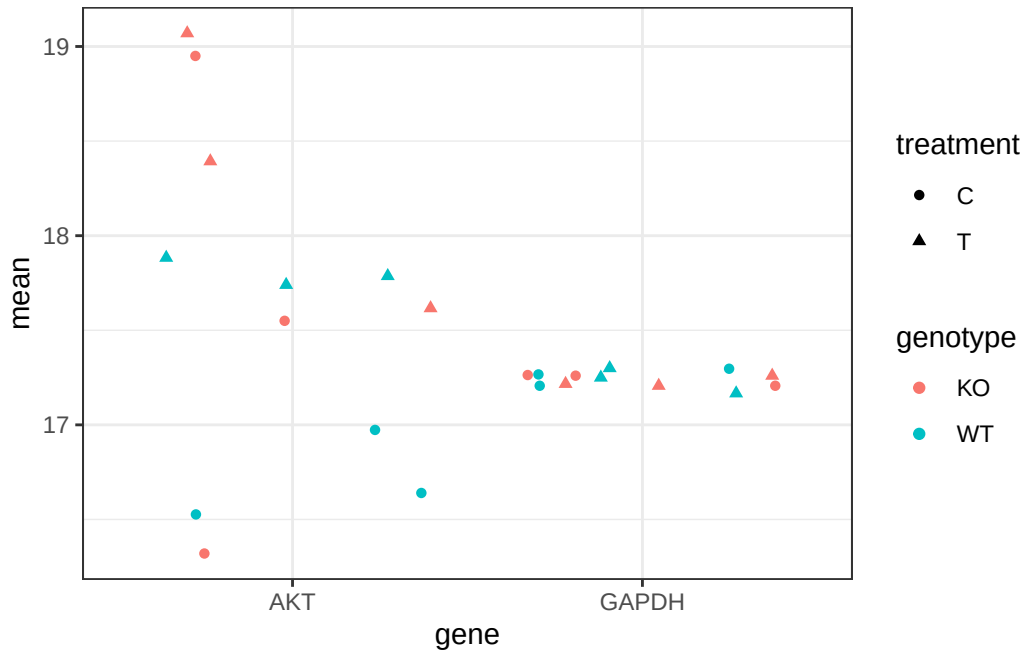
The idea with `ggplot2` is that one can add layers on your plot. You start with an empty canvas by calling the function `ggplot` and you specify its aesthetics within the function `aes`. Here we want to plot the cycle threshold by sample id, color by treatment and the shape will be the genotype.

[1] Check the appendix for more details.

```
p <- ggplot2::ggplot(
    summarised_qpcr,
    aes(x = gene, y = mean, color = genotype, shape = treatment)
)
```

To get the points for each group in a way they don't overlap with each other, we use the function `geom_jitter` from `ggplot2`. This function is useful when one of the axis is categorical, which in this case is the gene.

```
p +
    geom_jitter() +
    theme_bw()
```



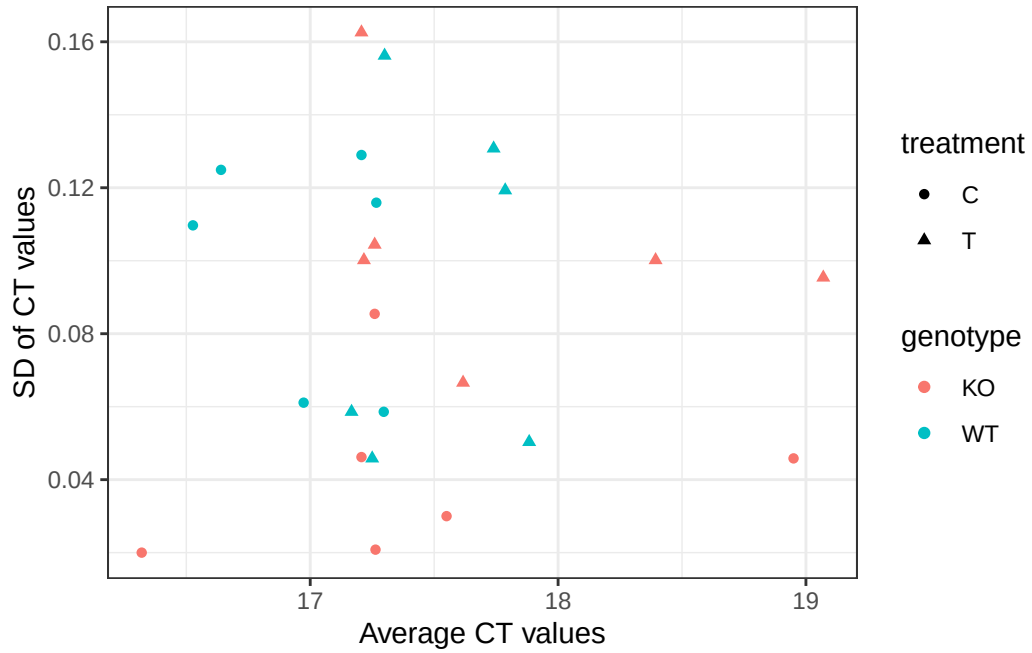
We can also modify the theme of the plot by using one of the functions `theme_*`. Here we use the `theme_bw`, all the basic options are here: https://ggplot2.tidyverse.org/reference/ggtheme.html.

Already from the average cycle threshold plot for each gene we can draw some conclusions. First we see that the loading worked very well for all samples, as the average CT is almost the same for all samples. Also from here we can see a difference in treatments among the WT samples from the AKT gene. Lastly, the control AKT gene from the KO genotype is highly variable.

5.2 Are there any outliers?

A common mistake is to use the data as it is. The exploratory analysis is very important as it can uncover outliers. One way to check the presence of weird samples is to plot the average CT values versus their standard deviation. The function `geom_point` gives a scatterplot. On the other hand, the function `labs` modifies the name of the axis, title and other things. Here we only change the x and y axis, but to change the title simply modify the argument `title` within the `labs` function.

```
ggplot(
    summarised_qpcr,
    aes(x = mean, y = sd, color = genotype, shape = treatment)
) +
    geom_point() +
    labs(
        x = "Average CT values",
        y = "SD of CT values"
    ) +
    theme_bw()
```



It seems that there is no outlier in this case. We can now proceed to the next chapter, in which we normalize the data.

Part III

Modeling

6 Normalizing the data

One crucial step is the delta CT normalization, in which we use the housekeeping genes to correct for the loading in the qPCR. In this small chapter we normalize the data and save it so it can be used during the modelling.

Before normalizing we would like to remove the technical samples that look like outliers. This is important because if there is a value that looks plainly wrong, it will affect the results.

And the last step before moving on to the next step is to once again plot the data. Even though we already explored the data once in the last chapter, plotting is never enough, it helps us understand what is going on with the experiments.

```
library(dplyr)
library(tidyr)

library(ggplot2)
```

```
qpcr <- readRDS("../checkpoints/cleaning/qpcr.rds")
summarised_qpcr <- readRDS("../checkpoints/preparing/summarised_qpcr.rds")
```

6.1 delta CT

The classical normalization method is the delta CT. We subtract the geometric average of the housekeeping genes from each average CT value of the genes of interest.

```
# specify the genes of interest and housekeeping genes, any
# number of genes can be used here. They are used in the code below
# during the normalization procedure
genes_of_interest <- c("AKT")
hk_genes <- c("GAPDH")

normalized_qpcr <- summarised_qpcr %>%
    group_by() %>%
    pivot_wider(
```

```
        id_cols = c("sample_id", "group", "genotype", "treatment"),
        values_from = "mean",
        names_from = "gene"
    ) %>%
    # apply the operation in each row individually
    rowwise() %>%
    mutate(hk_geom = exp(mean(log(!!sym(hk_genes))))) %>%
    # now we convert back to the long format but only for the
    # genes of interest, so we can apply for each row the difference
    pivot_longer(
        cols = all_of(genes_of_interest),
        names_to = "gene",
        values_to = "mean"
    ) %>%
    mutate(dct = mean - hk_geom)

normalized_qpcr %>% head
```

```
# A tibble: 6 x 9
  sample_id group genotype treatment GAPDH hk_geom gene   mean    dct
  <fct>     <chr> <chr>    <chr>     <dbl>   <dbl> <chr> <dbl>  <dbl>
1 20        WT C  WT       C          17.3    17.3 AKT    16.5 -0.740
2 21        WT C  WT       C          17.2    17.2 AKT    16.6 -0.567
3 22        WT C  WT       C          17.3    17.3 AKT    17.0 -0.323
4 23        WT T  WT       T          17.3    17.3 AKT    17.7  0.440
5 24        WT T  WT       T          17.2    17.3 AKT    17.8  0.537
6 25        WT T  WT       T          17.2    17.2 AKT    17.9  0.717
```

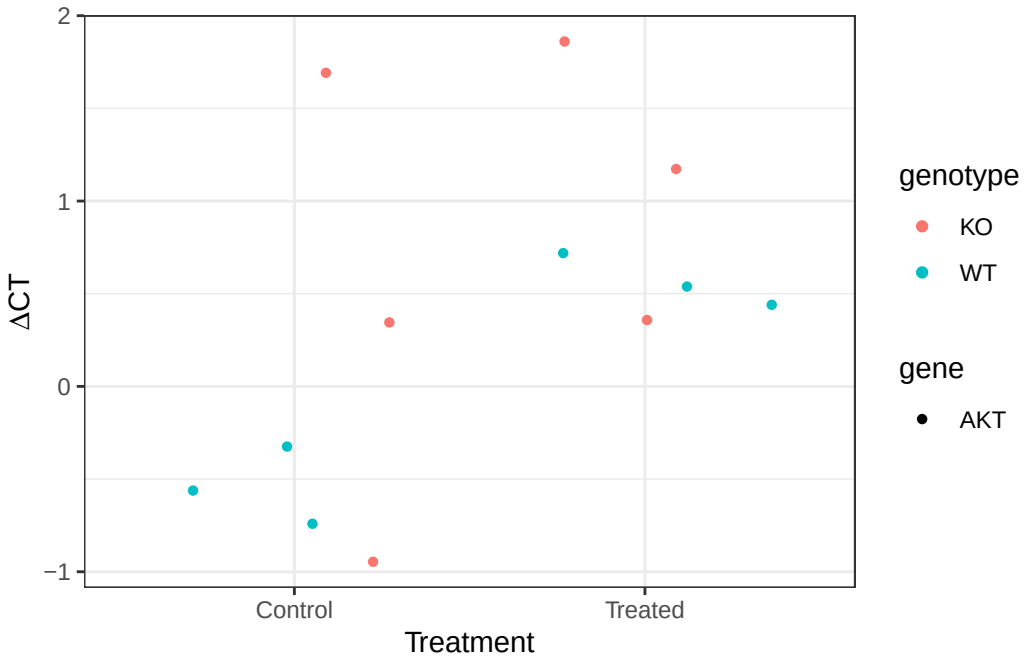
Notice here how important it is `pivot_longer` and `pivot_wider`. They allow us to perform several operations on our dataset.

6.2 Exploring normalized data

With the normalized dataset we can now investigate and compare the groups.

```
normalized_qpcr %>%
    # convert the C and T to control and treated
    mutate(treatment = ifelse(treatment == "C", "Control", "Treated")) %>%
    ggplot(aes(y = dct, x = treatment, color = genotype, shape = gene)) +
```

```
    geom_jitter() +
    labs(x = "Treatment", y = expression(Delta*"CT")) +
    theme_bw()
```

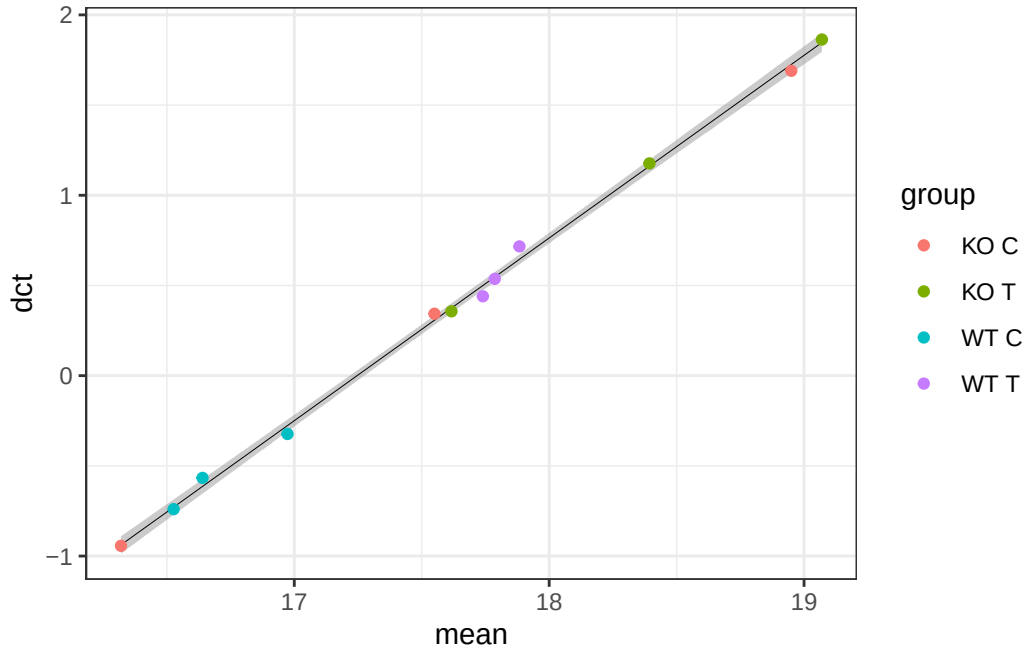


Already from this plot we see that in the control condition, the KO genotype is highly variable, so it is very unlikely we see any differences, in average between control and the treated condition. But that does not mean anything, it could be that your question is if the treatment stabilizes the KO condition of the gene.

And lastly we compare the CT values with the ΔCT values. It is another way to see how well the housekeeping genes are performing.

```
normalized_qpcr %>%
    ggplot(aes(x = mean, y = dct, color = group)) +
    geom_smooth(
        method = "lm",
        color = "black",
        alpha = 0.5,
        size = .1,
        formula = "y~x"
    ) +
    geom_point() +
```

```
  theme_bw()
```



6.3 Checkpoint

```
saveRDS(normalized_qpcr, "../checkpoints/normalizing/normalized_qpcr.rds")
```

7 Answering questions

We have been preparing and analysing our data to answer our questions. In this case we have two:

1. Is there any difference between treatment and control in the wild type (WT) condition?
2. Does the effect of treatment depends on genotype?

The questions will be answered in the next section by using the framework of linear regression. Linear regression is doing almost the same thing as a t-test. Also if you want to test normality of the values this is possible, but since the sample size is very low it is very difficult to reject the null hypothesis that the samples do not follow a normal distribution.

The regression framework allows us to extend it to other uses, such as using other types of distributions. One example is when we are dealing with proportions or percentages, usually one would use the gamma distribution instead of the normal distribution to model the data.

Let's move to answering the questions!

7.1 First question

Let us recap the question here:

> Is there any difference between treatment and control in the wild type (WT) condition?

For this we will need to load once again the packages and our previous checkpoint. R already provides in the base package the function `lm` to perform linear regression, so this is actually not necessary to load.

```
library(dplyr)
```

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

term	estimate	std.error	statistic	p.value
(Intercept)	-0.5433333	0.1028963	-5.280396	0.0061685
groupWT T	1.1077778	0.1455174	7.612684	0.0015982

```
The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
library(broom)

# we load only the normalized data as this is what will be used when doing
# the comparisons
normalized_qpcr <- readRDS("../checkpoints/normalizing/normalized_qpcr.rds")
```

The code below executes the comparison and prints the output. What we want to read from the linear regression output is the estimate of the slope, that will represent the difference between the two groups! One nice thing of using linear regression is that you get the difference values between your conditions, and that might be used to further interpret the results.

```
analysis_first_question <- normalized_qpcr %>%
    # first filter just for the WT
    dplyr::filter(grepl("WT", group)) %>%
    # Now we factor so the first level is control and the difference
    # then represents treatment - control.
    dplyr::mutate(group = factor(group, levels = c("WT C", "WT T"))) %>%
    # perform the linear regression
    lm(dct ~ group, data = .) %>%
    # tidy up the results so we can display
    broom::tidy()


analysis_first_question %>%
    kableExtra::kbl() %>%
    kableExtra::kable_classic(full_width = FALSE)
```

To calculate the fold change one needs to flip the sign for the estimate, since an increase of CT is actually a decrease in expression. The total fold change is 0.464008205171193 and log fold change is -1.10777777777778.

The way to calculate the fold change is $2^{(-\text{estimate})}$, where estimate corresponds to the second row and second column of the table above. And the log fold change is just the negative value of estimate.

The conclusion for this question is that there is indeed a decrease in AKT expression levels in the treated condition for the WT genotype. We had already seen this from the plots of the previous chapters.

I would recommend to report this values along with the previous plots, as the CT values are also an important aspect of the analysis, as an increase of 30 to 31 is not the same as 16 to 17.

7.2 Second question

We now move on to the second question.

> Does the effect of treatment depends on genotype?

Here we are interested in the interaction between the genotype and treatment. R has an special way to encode this when doing linear regression, by using the star symbol *.

```
analysis_second_question <- normalized_qpcr %>%
    dplyr::mutate(
        group = factor(group, levels = c("WT C", "WT T", "KO C", "KO T")),
        genotype = factor(genotype, levels = c("WT", "KO"))
    ) %>%
    # notice here the star. we are now interested in the interaction
    lm(dct ~ genotype * treatment, data = .) %>%
    broom::tidy()

analysis_second_question %>%
    kableExtra::kbl() %>%
    kableExtra::kable_classic(full_width = FALSE)
```

We had already seen that actually in the knockout condition there was a lot of variability. This is reflected in the results here. The last line of the table above presents the result that interest us, the interaction between treatment and genotype. We see that the p-value is around 0.71, so we don't have enough evidence to reject the null hypothesis, which is there is no effect in the genotype and treatment.

term	estimate	std.error	statistic	p.value
(Intercept)	-0.5433333	0.4440748	-1.2235175	0.2559465
genotypeKO	0.9066667	0.6280167	1.4436984	0.1868167
treatmentT	1.1077778	0.6280167	1.7639305	0.1157518
genotypeKO:treatmentT	-0.3388889	0.8881497	-0.3815673	0.7127173

8 Conclusion

Here we saw how to perform basic statistical analysis of the generated qPCR data and how to interpret it. The analysis does not stop here, it is important to also document it. In the next chapter we show a resource that will teach you how to generate such report and how you can customize it.

Part IV

Wrap up

9 Generating a report

Usually an analysis is done multiple times, so it is important we document what we did. We might come back in the future to add some more data points or we just want to know what happened when the analysis was performed.

Quarto markdown is a way to do so. By using quarto markdown you can mix both code and text together, so your analysis can be documented and reused in the future. As a matter of fact, this book was done in quarto.

The Posit team (Rstudio and tidyverse developers) have written amazing tutorials on how to use quarto markdown and author documents. The link below leads to a get started

https://quarto.org/docs/get-started/

Essentially the idea is that you need to know a bit of R and a bit of markdown. Markdown is an intuitive way of writing text. The paradigm is a bit different than that of Microsoft Word as you need to render your documents to get the final output.