

V. Statistical analysis in R (presentation)

Data Science Lab, University of Copenhagen

2026-05-06

Table of contents

Structure of a statistical analysis in R	1
Example A: Analysis of variance	2
Example B: Simple and multiple linear regression	12
Use of pipe operator	21
Outlook: Other analyses	22

Structure of a statistical analysis in R

The very basic structure of an R script doing a classical statistical analysis is as follows:

1. Load packages that you will be using.
2. Read the dataset to be analyzed. Possibly also do some data cleaning and manipulation.
3. Visualize the dataset by graphics and other descriptive statistics.
4. Fit and validate a statistical model.
5. Hypothesis testing. Possibly also post hoc testing.
6. Visualize and/or report: model parameters and/or model predictions.

Of course there are variants of this set-up, and in practice there will often be some iterations of the steps.

In this manuscript we will exemplify the proposed steps in the analysis of two simple datasets:

- A) One-way ANOVA of the expression of the IGFL4 gene against the skin type in psoriasis patients.
- B) Multiple linear regression of volume of cherry trees against their diameter and height.

Step 1: Load packages

For both examples we will use `ggplot2` to make plots, and to be prepared for data manipulations we simply load this together with the rest of the `tidyverse`. Moreover, we will also use extra plotting functionalities from the `ggfortify` and `GGally` packages.

The psoriasis data are provided in an Excel sheet, so we also load `readxl`. Finally, we will use the package `emmeans` to make post hoc tests and to report model predictions.

Remember that you should install the wanted packages before they can be used (but you only need to install the packages once!) To take full advantage of the `emmeans` package you should also install the packages `multcomp` and `multcompView` packages.

Thus,

```
#install.packages("tidyverse")
#install.packages("ggfortify")
#install.packages("GGally")
#install.packages("readxl")
#install.packages("emmeans")
#install.packages("multcomp")
#install.packages("multcompView")
#install.packages("lme4")
library(tidyverse)
library(ggfortify)
library(GGally)
library(readxl)
library(emmeans)
library(multcomp)
library(multcompView)
library(lme4)
```

Now we have done step 1 for both analyses. Next we will look specifically at the two examples. Finally, we conclude with a brief outlook on other statistical models in R.

Example A: Analysis of variance

Step 2: Data

Psoriasis is an immune-mediated disease that affects the skin. Researchers carried out an micro-array experiment with skin from 37 people in order to examine a potential association between the disease and a certain gene (IGFL4). For each of the 37 samples the gene expression was measured as an intensity. Fifteen skin samples were from psoriasis patients and from a part of the body affected by the disease (`psor`); 15 samples were from psoriasis patients but from

a part of the body not affected by the disease (**psne**); and 7 skin samples were from healthy people (**control**).

The data are saved in the file **psoriasis.xlsx**. At first the variable **type** is stored as a character variable we change it to a factor (and check that indeed there are 15, 15 and 7 patients in the three groups).

```
psorData <- read_excel("psoriasis.xlsx")
psorData
```

```
# A tibble: 37 x 3
  type    intensity typeNum
  <chr>    <dbl>    <dbl>
1 psne      0.841      1
2 psne      0.955      1
3 psne      1.07      1
4 psne      1.11      1
5 psne      1.18      1
6 psne      1.20      1
7 psne      1.32      1
8 psne      1.30      1
9 psne      1.39      1
10 psne     1.41      1
# i 27 more rows
```

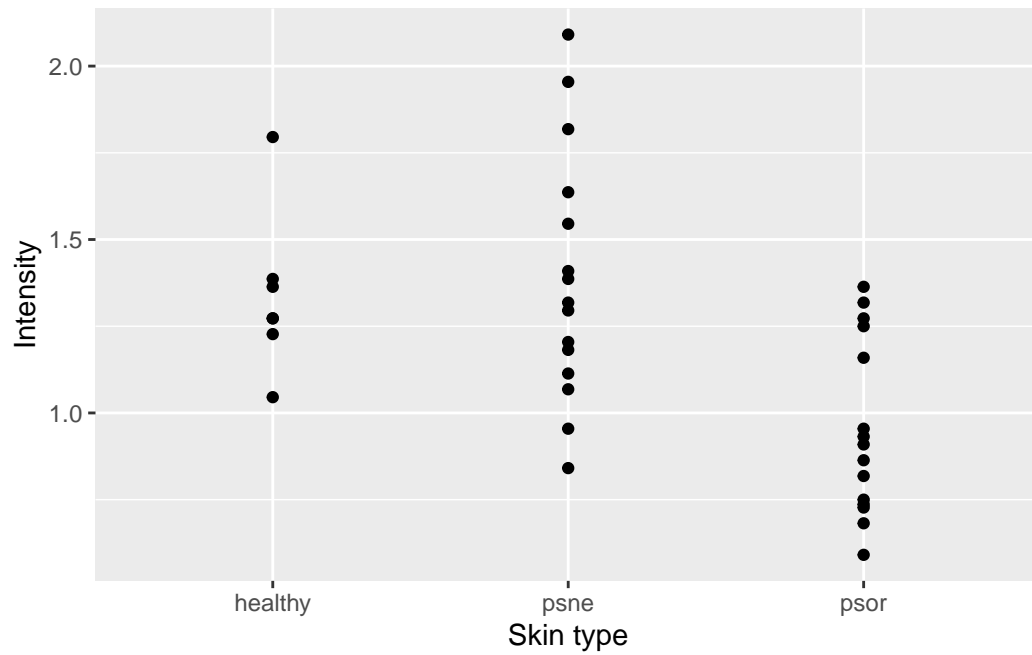
```
psorData <- mutate(psorData, type = factor(type))
count(psorData, type)
```

```
# A tibble: 3 x 2
  type      n
  <fct> <int>
1 healthy    7
2 psne     15
3 psor     15
```

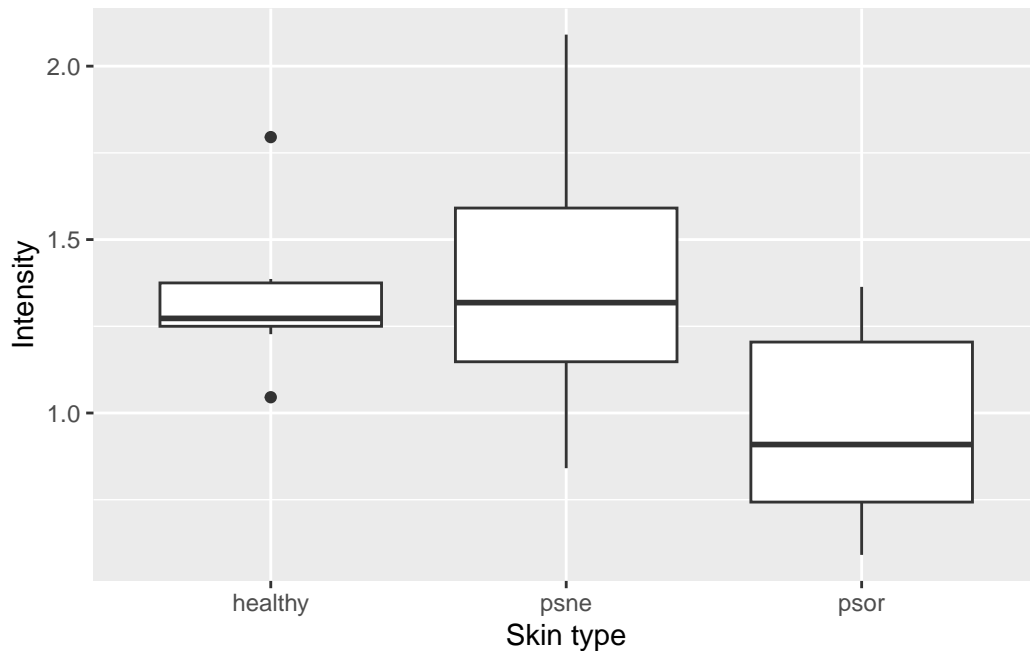
Step 3: Descriptive plots and statistics

To get an impression of the data we make two plots and compute group-wise means and standard deviations.

```
ggplot(psorData, aes(x=type, y=intensity)) +
  geom_point() +
  labs(x="Skin type", y="Intensity")
```



```
ggplot(psorData, aes(x=type, y=intensity)) +
  geom_boxplot() +
  labs(x="Skin type", y="Intensity")
```



```
psorData %>%
  group_by(type) %>%
  summarise(avg=mean(intensity), sd=sd(intensity))
```

```
# A tibble: 3 x 3
  type      avg    sd
  <fct>   <dbl> <dbl>
1 healthy 1.34  0.230
2 psne    1.39  0.363
3 psor    0.955 0.255
```

Step 4: Fit of oneway ANOVA, model validation

The scientific question is whether the gene expression level differs between the three types/groups. Thus, the natural type of analysis is a oneway analysis of variance (ANOVA). The oneway ANOVA is fitted with `lm`. It is a good approach to assign a name (below *oneway*) to the object with the fitted model. This object contains all relevant information and may be used for subsequent analysis. Note that we have logarithmic transformed the response as intensities often live on a multiplicative scale.

```
oneway <- lm(log(intensity) ~ type, data=psorData)
oneway
```

Call:

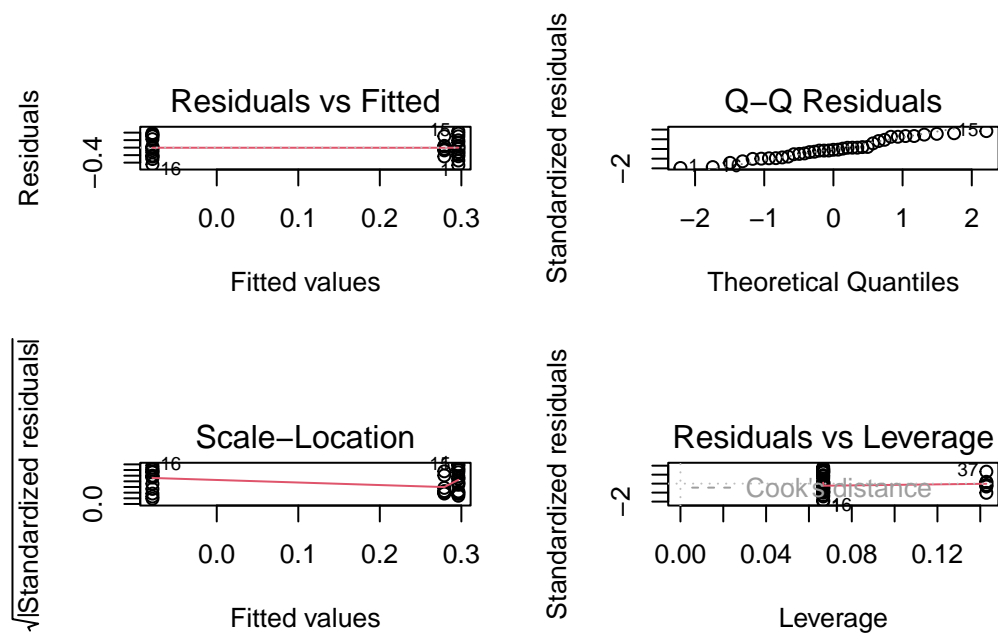
```
lm(formula = log(intensity) ~ type, data = psorData)
```

Coefficients:

(Intercept)	typepsne	typepsor
0.27910	0.01724	-0.35794

Mathematically this model makes assumptions about linearity (which, however, will always be satisfied for a oneway ANOVA – why?), variance homogeneity, and normality. These assumptions are checked visually via four plots generated from the model residuals. In R it is easy to make these plots; simply apply `plot` on the `lm`-object.

```
par(mfrow=c(2,2)) # makes room for 4=2x2 plots!  
plot(oneway)
```



The model validation plots do not show any evident problems.

Remark: Using the `autoplot()` function from the `ggfortify` package you can also make a ggplot-version of these plots. This can be aesthetically more pleasing, you avoid calling the `par()` function, and there is more room for the plot on a small laptop screen. But mathematically it is the same plots that will be produced!

```
autoplot(oneway)
```

Warning: `fortify(<lm>)` was deprecated in ggplot2 4.0.0.

i Please use `broom::augment(<lm>)` instead.

i The deprecated feature was likely used in the ggfortify package.

Please report the issue at <<https://github.com/sinhrks/ggfortify/issues>>.

Warning: `aes_string()` was deprecated in ggplot2 3.0.0.

i Please use tidy evaluation idioms with `aes()`.

i See also `vignette("ggplot2-in-packages")` for more information.

i The deprecated feature was likely used in the ggfortify package.

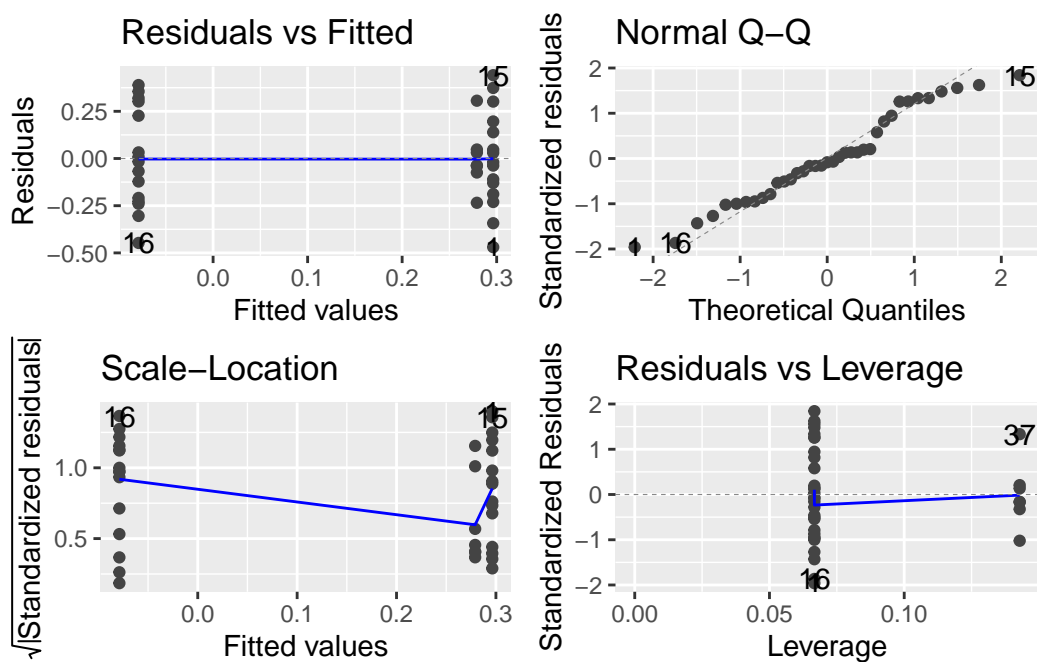
Please report the issue at <<https://github.com/sinhrks/ggfortify/issues>>.

Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.

i Please use `linewidth` instead.

i The deprecated feature was likely used in the ggfortify package.

Please report the issue at <<https://github.com/sinhrks/ggfortify/issues>>.



Step 5: Hypothesis test + Post hoc tests

It is standard to carry out an F -test for the overall effect of the explanatory variable. To be precise, the hypothesis is that the expected values are the same in *all* groups. The most easy way to do this test is to use `drop1`. The option `test="F"` is needed to get the F -test:

```
drop1(oneway, test="F")
```

Single term deletions

Model:

```
log(intensity) ~ type
      Df Sum of Sq    RSS      AIC F value    Pr(>F)
<none>                2.0924 -100.286
type    2      1.2204 3.3128  -87.286   9.9153 0.0004052 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Thus, the overall test for homogeneity between the groups show, that the groups are not all the same. But it might be that the gene expression in two of the three groups, say, are not significantly different. To investigate that we do post hoc testing. This is nicely done within the framework of *estimated marginal means* using the `emmeans` package. Here `emmeans` makes the estimated marginal means (that is the predicted gene expression intensity on the log scale), and the `pairs` command provide post hoc pairwise comparisons:

```
emmeans(oneway, ~type)
```

type	emmean	SE	df	lower.CL	upper.CL
healthy	0.2791	0.0938	34	0.0885	0.4696
psne	0.2963	0.0641	34	0.1662	0.4265
psor	-0.0788	0.0641	34	-0.2090	0.0513

Results are given on the log (not the response) scale.
Confidence level used: 0.95

```
pairs(emmeans(oneway, ~type))
```

contrast	estimate	SE	df	t.ratio	p.value
healthy - psne	-0.0172	0.1140	34	-0.152	0.9874
healthy - psor	0.3579	0.1140	34	3.152	0.0092
psne - psor	0.3752	0.0906	34	4.142	0.0006

Results are given on the log (not the response) scale.

P value adjustment: tukey method for comparing a family of 3 estimates

It can be convenient to summarize the pairwise comparisons in the so-called *compact letter display*. Here a grouping is performed such that groups *not sharing* a letter are significantly different. Previous versions of the **emmeans** package had a function for doing this (called CLD). However, the author of the **emmeans** package does not approve of this way of displaying the results, and consequently he has removed the CLD function from the package! But instead you can use the `cld` function from the **multcomp** package. Notice that both the **multcomp** and the **multcompView** package must be installed.

Using the syntax `multcomp::cld` we may avoid loading the **multcomp** package. Long story short; here is the code:

```
cld(emmeans(oneway, ~type))
```

type	emmean	SE	df	lower.CL	upper.CL	.group
psor	-0.0788	0.0641	34	-0.2090	0.0513	1
healthy	0.2791	0.0938	34	0.0885	0.4696	2
psne	0.2963	0.0641	34	0.1662	0.4265	2

Results are given on the log (not the response) scale.

Confidence level used: 0.95

P value adjustment: tukey method for comparing a family of 3 estimates

significance level used: alpha = 0.05

NOTE: If two or more means share the same grouping symbol,
then we cannot show them to be different.
But we also did not show them to be the same.

We see that the “grouping letters” appearing in the column `.group` actually are numbers. But the interpretation is the same. Namely that `psor` (letter=1) is significantly different from both `healthy` (letter=2) and `psne` (letter=2), whereas that two latter are not significantly different (they share the letter “2”). Note: If you want actual letters instead of digits, then add the option `Letters=letters`.

Or restated in biological terms: The groups *healthy* and *psne* appears to have the same expression of the IGFL4 gene, and this expression is significantly larger than in the *psor* group. Biologically this is consistent with the IGFL4 gene being related to psoriasis as *psne* corresponds to skin not affected by the disease.

Step 6: Report of model parameters

In this example it is natural to report the estimated intensities as well as the comparisons between the 3 groups. As the analysis was done using a log transformation it is also advisable to backtransform. Both things can be done automatically by the **emmeans** package, where the option `type="response"` (here `type` is the name of an option inside the R command, and it is a coincidence that it has the same name as the variable *type*) requests the backtransformation:

```
emmeans(oneway, ~type, type="response")
```

type	response	SE	df	lower.CL	upper.CL
healthy	1.322	0.1240	34	1.093	1.60
psne	1.345	0.0861	34	1.181	1.53
psor	0.924	0.0592	34	0.811	1.05

Confidence level used: 0.95

Intervals are back-transformed from the log scale

```
confint(pairs(emmeans(oneway, ~type, type="response")))
```

contrast	ratio	SE	df	lower.CL	upper.CL
healthy / psne	0.983	0.112	34	0.744	1.30
healthy / psor	1.430	0.162	34	1.083	1.89
psne / psor	1.455	0.132	34	1.166	1.82

Confidence level used: 0.95

Conf-level adjustment: tukey method for comparing a family of 3 estimates

Intervals are back-transformed from the log scale

Although we don't recommend usage of standard errors in the context of backtransformed parameters, we see that a standard error is provided on the backtransformed parameters.

Iteration of step 3 to step 6

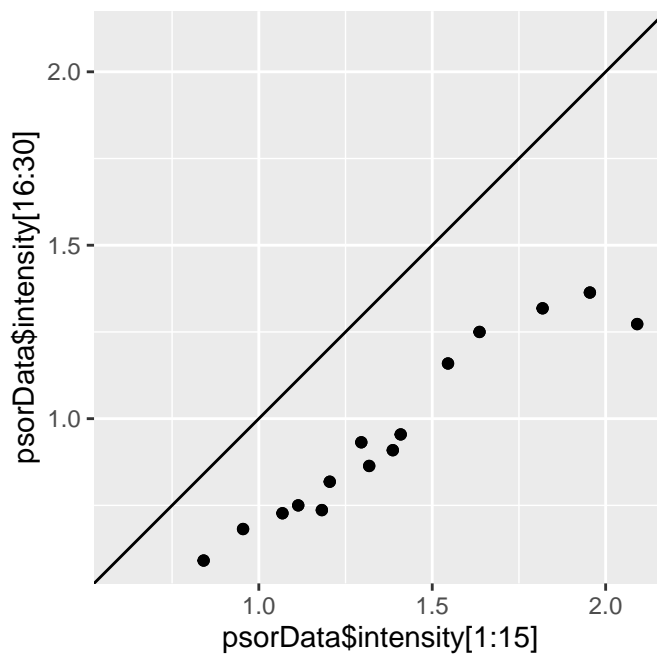
Actually the used statistical model is wrong! At least there appears to be a strong correlation between the *n*'th observations of *psne* of *psor*. This is evident from the following two descriptive statistics:

```
# correlation  
cor(psorData$intensity[1:15], psorData$intensity[16:30])
```

```
[1] 0.9609382
```

```
# scatter plot
qplot(x=psorData$intensity[1:15], y=psorData$intensity[16:30]) +
  geom_point() +
  geom_abline(intercept=0,slope=1) +
  coord_equal(xlim=c(0.6,2.1),ylim=c(0.6,2.1))
```

Warning: `qplot()` was deprecated in ggplot2 3.4.0.



What is going on here?

The first 30 observations presumably do not come from 30 different people, but are rather two measurements from each of 15 people, one from an affected skin area and one from non-affected area. In that case, the correct analysis should include a random effect of person. First we introduce the apparently missing variable in the dataset:

```
psorData$Id <- factor(c(1:15,1:15,16:22))
```

Then we can do the correct mixed effects model using the `lmer()` function from the **lme4** package:

```
oneway_with_random_effect <- lmer(log(intensity) ~ type + (1|Id), data=psorData)
```

The syntax used in the other steps now change a bit at some places due to the usage of the `lme4` package. We will not do all the steps here, but for comparison let's check that the preferred analysis including the random effect indeed is more powerful:

```
drop1(oneway, test="F")
```

Single term deletions

```
Model:
log(intensity) ~ type
      Df Sum of Sq  RSS      AIC F value    Pr(>F)
<none>                2.0924 -100.286
type    2    1.2204 3.3128  -87.286   9.9153 0.0004052 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
drop1(oneway_with_random_effect, test="Chisq")
```

Single term deletions

```
Model:
log(intensity) ~ type + (1 | Id)
      npar      AIC    LRT   Pr(Chi)
<none>      -34.542
type        2  20.442 58.984 1.555e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Example B: Simple and multiple linear regression

Step 2: Data

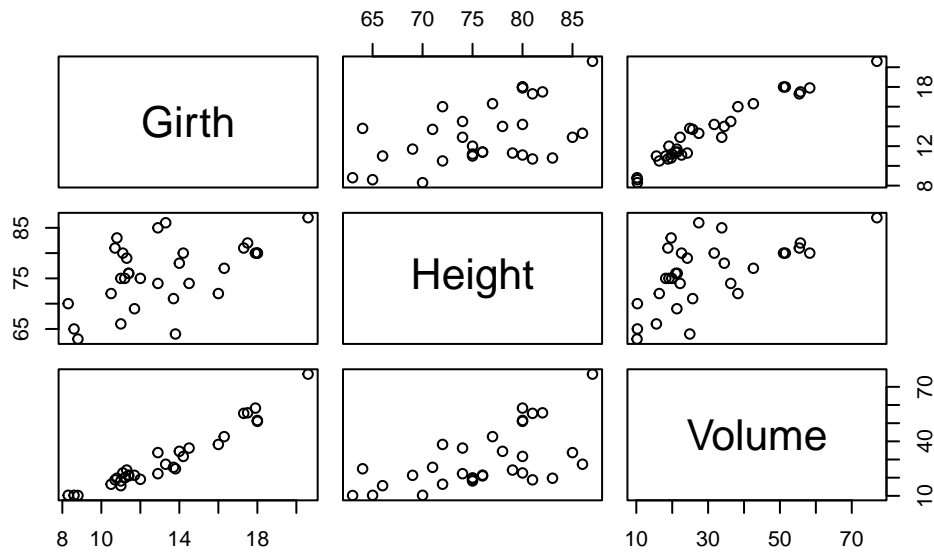
For the first part of the presentation we will use data from 31 cherry trees available in the built in dataset `trees`: Girth, height and tree volume have been measured for each tree. Please note, that the help page `?trees` states, that girth is the tree diameter. Interest is in the association between tree volume on the one side and girth and height on the other side.

```
data(trees)
```

Step 3: Visualization of raw data

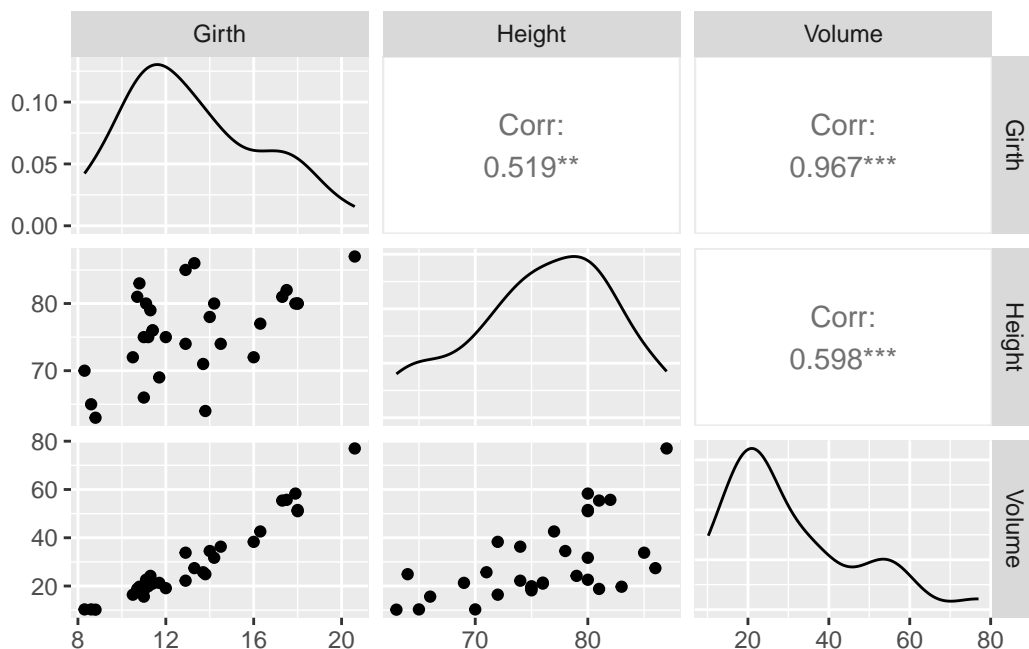
We start by plotting the data. If you plot a data frame, then you get a display of pairwise scatter plots (using Base R graphics):

```
plot(trees)
```



A more fancy version of this is available in **GGally**, where the diagonal is used to visualize the marginal distribution of the three tree variables:

```
ggpairs(trees)
```



Step 4: Fitting and validating a simple linear regression model

We start out with a simple linear regression with volume as outcome and girth as covariate. This model is fitted with the `lm` function. It is a good approach to assign a name (below *linreg1*) to the object with the fitted model. This object contains all relevant information and may be used for subsequent analysis.

```
linreg1 <- lm(Volume~Girth, data=trees)
linreg1
```

Call:

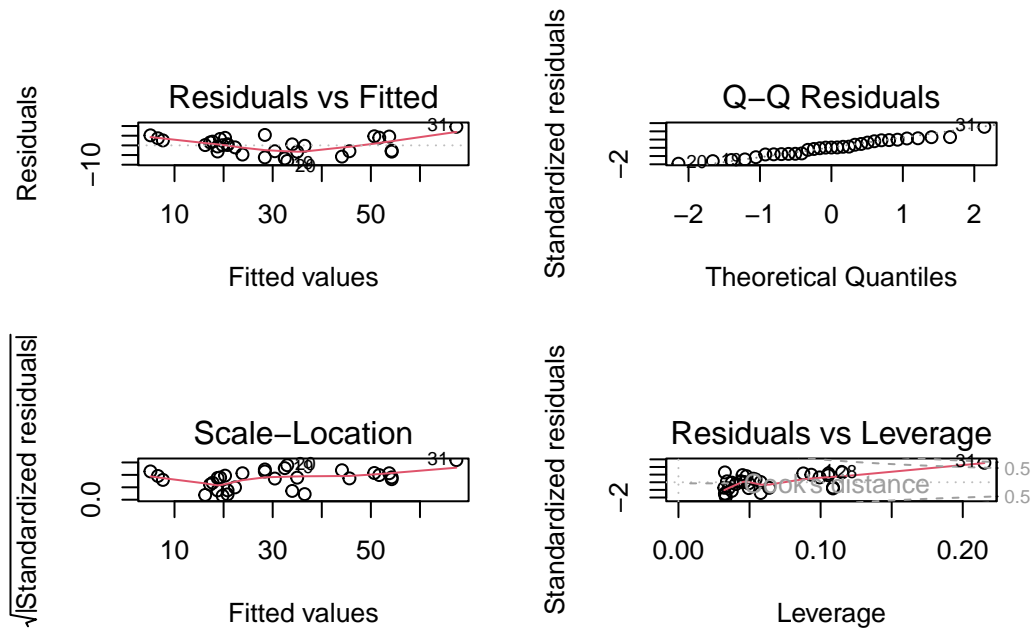
```
lm(formula = Volume ~ Girth, data = trees)
```

Coefficients:

```
(Intercept)      Girth
   -36.943      5.066
```

Mathematically this model makes assumptions about linearity, variance homogeneity, and normality. These assumptions are checked visually via four plots generated from the model residuals. In R it is easy to make these plots; simply apply `plot` on the `lm`-object.

```
par(mfrow=c(2,2)) # makes room for 4=2x2 plots!
plot(linreg1)
```

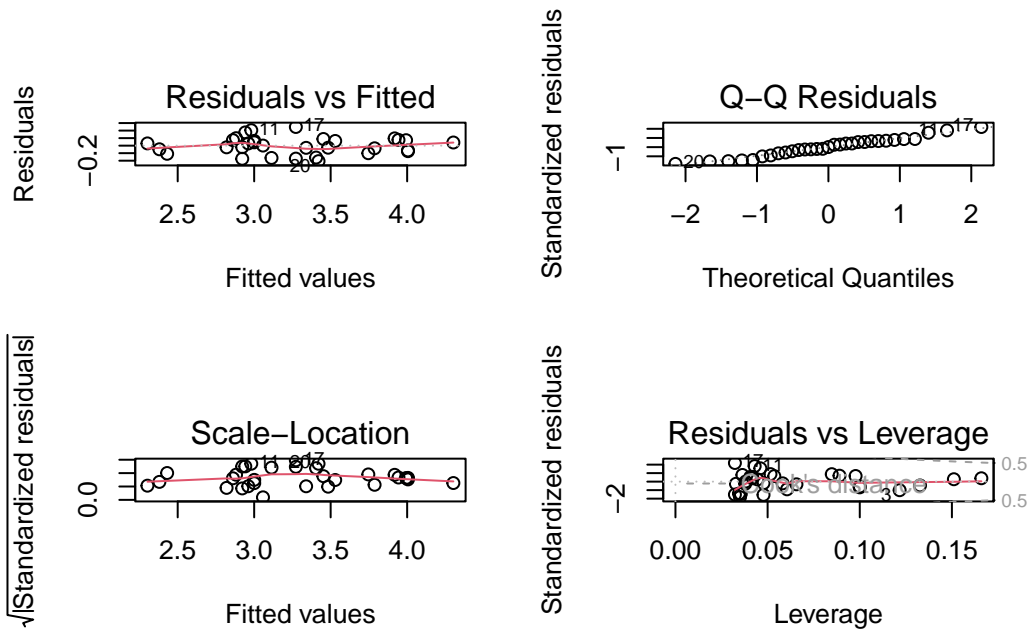


In this case there appears to be problems with the model. In particular, the upper left plot shows a quadratic tendency suggesting that the linearity assumption is not appropriate. Perhaps a data transformation can help us out.

Step 4 iterated: Transformation

The linear regression model above says that an increment of Δ_g of girth corresponds to an increment of volume by $\beta\Delta_g$ where β is the slope parameter, i.e., interpretation is concerned with *absolute changes*. Perhaps it is more reasonable to talk about *relative changes*: An increment of girth by a factor c_g corresponds to an increment of volume of γc_g . This corresponds to a linear regression model on the log-transformed variables, which is easily fitted. Notice that `log` in R is the natural logarithm.

```
linreg2 <- lm(log(Volume) ~ log(Girth), data=trees)
par(mfrow=c(2,2))
plot(linreg2)
```



The statistical validity of this model is much better! Which is nice as it also has a much more meaningful biological/physical interpretation!

Step 6: Report of the model

Parameter estimates, standard errors are computed and certain hypothesis tests are carried out by the `summary()` function:

```
summary(linreg2)
```

Call:

```
lm(formula = log(Volume) ~ log(Girth), data = trees)
```

Residuals:

Min	1Q	Median	3Q	Max
-0.205999	-0.068702	0.001011	0.072585	0.247963

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-2.35332	0.23066	-10.20	4.18e-11 ***
log(Girth)	2.19997	0.08983	24.49	< 2e-16 ***

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.115 on 29 degrees of freedom
```

```
Multiple R-squared:  0.9539,    Adjusted R-squared:  0.9523
```

```
F-statistic: 599.7 on 1 and 29 DF,  p-value: < 2.2e-16
```

The coefficient table is the most important part of the output. It has two lines — one for each parameter in the model — and four columns. The first line is about the intercept, the second line is about the slope. The columns are

- **Estimate:** The estimated value of the parameter (intercept or slope).
- **Std. Error:** The standard error (estimated standard deviation) associated with the estimate in the first column.
- **t value:** The t -test statistic for the hypothesis that the corresponding parameter is zero. Computed as the estimate divided by the standard error.
- **Pr(>|t|):** The p -value associated with the hypothesis just mentioned. In particular the p -value in the second line is for the hypothesis that there is no effect of girth on volume.

Below the coefficient table you find, among others, the **Residual standard error**, i.e., the estimated standard deviation for the observations.

If you prefer to report confidence intervals for the parameters (intercept and slope) instead of standard error then use `confint()`:

```
confint(linreg2)
```

```
                2.5 %    97.5 %  
(Intercept) -2.825083 -1.881566  
log(Girth)   2.016238  2.383702
```

Finally, don't forget to make the mathematical interpretation of the model and its parameter estimates! Thus, the model postulates the relation

$$\ln V \approx \alpha + \beta \ln G$$

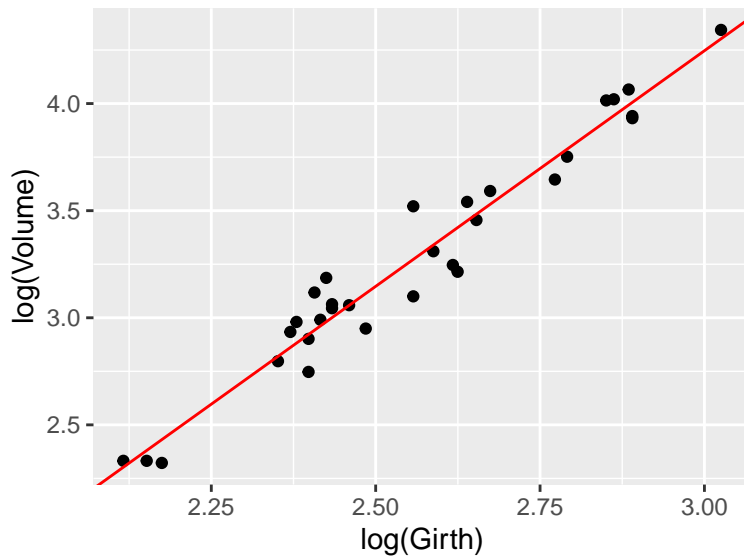
with estimates $\hat{\alpha} = -2.353$ and $\hat{\beta} = 2.200$. This corresponds to $V \approx e^{\alpha} \cdot G^{\beta}$, and we see that an increment of girth by 10%, say, corresponds to an increment of volume by a factor of $1.1^{2.2} = 1.23$, that is, by 23%.

Step 6: Visualization of the model

An excellent synthesis of a statistical analysis is often provided by a plot combining the raw data with the model fit.

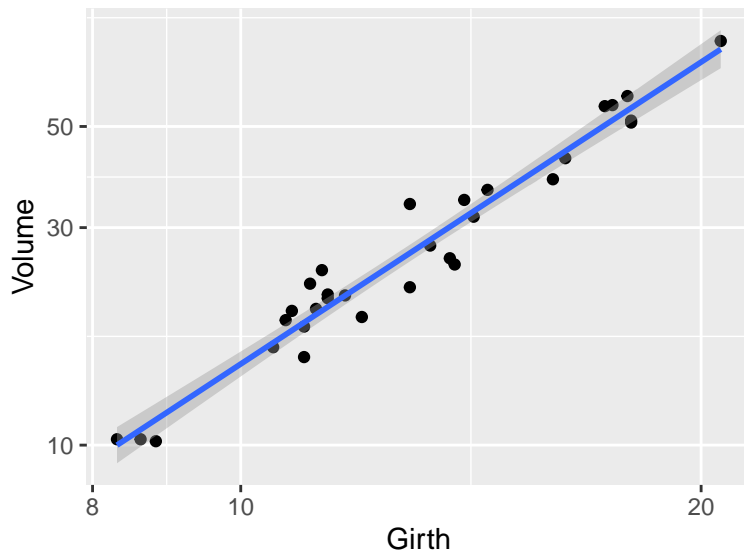
The estimated regression line can be added to a scatter plot in different ways (and with/without corresponding standard error curves):

```
ggplot(trees, aes(x=log(Girth), y =log(Volume))) +  
  geom_point() +  
  geom_abline(intercept=-2.35332, slope=2.19997, col="red")
```



```
ggplot(trees, aes(x=Girth, y=Volume)) +  
  geom_point() +  
  geom_smooth(method="lm", se=TRUE) +  
  scale_x_log10() +  
  scale_y_log10()
```

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



Step 4 iterated: Multiple linear regression

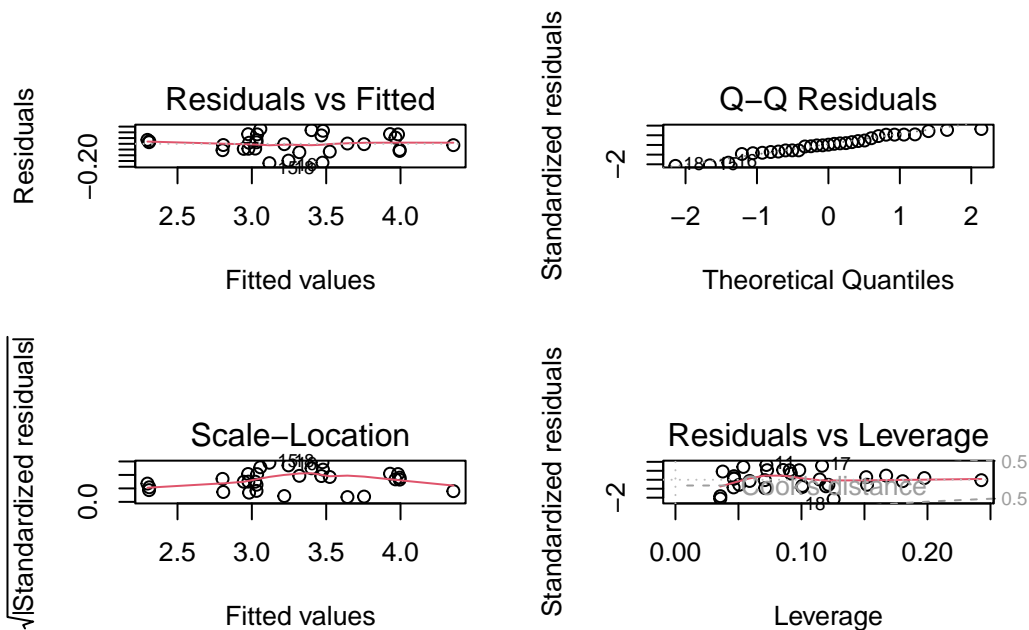
Several covariates can be included in a regression model, corresponding to multiple linear regression; simply write a plus between the covariates. Here is fitting of a model with log-girth as well as log-height included as explanatory variables:

```
linreg3 <- lm(log(Volume) ~ log(Girth) + log(Height), data=trees)
summary(linreg3)$coefficients
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-6.631617	0.79978973	-8.291701	5.057138e-09
log(Girth)	1.982650	0.07501061	26.431592	2.422550e-21
log(Height)	1.117123	0.20443706	5.464388	7.805278e-06

Don't forget to do model validation:

```
par(mfrow=c(2,2))
plot(linreg3)
```



Step 5: Hypothesis test

The F -test for comparison of two nested models may be carried out by the `anova` function. For example, `linreg3` is nested in `linreg2`, and the comparison between the two corresponds to the hypothesis that there is no extra information in height when girth is included (on log-scale).

```
anova(linreg3, linreg2)
```

Analysis of Variance Table

Model 1: $\log(\text{Volume}) \sim \log(\text{Girth}) + \log(\text{Height})$

Model 2: $\log(\text{Volume}) \sim \log(\text{Girth})$

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	28	0.18546				
2	29	0.38324	-1	-0.19778	29.86	7.805e-06 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

However, as an alternative to `anova` it is more simple to use the `drop1` command, which makes all the tests corresponding to removal of one of the explanatory variables. If you want p -values, then ask for F -tests:

```
drop1(linreg3, test="F")
```

Single term deletions

Model:

```
log(Volume) ~ log(Girth) + log(Height)
              Df Sum of Sq    RSS      AIC F value    Pr(>F)
<none>                        0.1855 -152.685
log(Girth)    1     4.6275 4.8130  -53.743   698.63 < 2.2e-16 ***
log(Height)   1     0.1978 0.3832 -132.185    29.86 7.805e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We see that `anova` and `drop1` give the same p-value ($p = 7.8 \times 10^{-6}$) for the effect of `log(Height)`. And in a Gaussian model (like this) this coincides with the p-value from the *t*-test for the hypothesis that there is no effect of a single numerical covariate. Thus,

```
summary(linreg3)$coefficients
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-6.631617	0.79978973	-8.291701	5.057138e-09
log(Girth)	1.982650	0.07501061	26.431592	2.422550e-21
log(Height)	1.117123	0.20443706	5.464388	7.805278e-06

So should you do all three tests? Definitely not!

- In general it is not recommendable to use `summary` for doing tests: The *F*-tests and *t*-tests only coincide for hypothesis on a single parameter. And you may also be deceived by the model parametrizations when interpreting *t*-tests.
- To use `drop1` you only need to fit the large model (here *linreg3*). Thus, it gives shorter and more easily readable R code.

Use of pipe operator

Although we did not use it systematically in the above, the pipe operator may increase readability of the code. For example, the two commands below do exactly the same:

```
confint(pairs(emmeans(oneway, ~type, type="response")))
```

contrast	ratio	SE	df	lower.CL	upper.CL
healthy / psne	0.983	0.112	34	0.744	1.30
healthy / psor	1.430	0.162	34	1.083	1.89
psne / psor	1.455	0.132	34	1.166	1.82

Confidence level used: 0.95

Conf-level adjustment: tukey method for comparing a family of 3 estimates

Intervals are back-transformed from the log scale

```
oneway %>% emmeans(~type, type="response") %>% pairs() %>% confint()
```

contrast	ratio	SE	df	lower.CL	upper.CL
healthy / psne	0.983	0.112	34	0.744	1.30
healthy / psor	1.430	0.162	34	1.083	1.89
psne / psor	1.455	0.132	34	1.166	1.82

Confidence level used: 0.95

Conf-level adjustment: tukey method for comparing a family of 3 estimates

Intervals are back-transformed from the log scale

Outlook: Other analyses

The `lm` function is used for linear models, that is, models where data points are assumed to be independent with a Gaussian distribution (and typically also with the same variance). Obviously this class of models is not always appropriate, and there exists functions for many, many more situations and data types. Here we just mention a few functions corresponding to common data types and statistical problems.

- `glm()`: For independent, but non-Gaussian data. Examples are binary outcomes (logistic regression) and outcomes that are counts (Poisson regression). `glm` is short for generalized linear models, and the `glm()` function is part of the base installation of R. Remark: SAS users should not confuse this with **PROC GLM**, which does linear normal models like `lm()`.
- `lmer()` and `glmer()`: For data with dependence structures that can be described by random effects, e.g., block designs. `lme` is short for linear mixed effects (Gaussian data), `glmer` is short for generalized linear mixed effects (binary or count data). Both functions are part of the **lme4** package. Actually, we did touch upon a mixed effects model above.
- `nls()`: For non-linear regression, e.g., dose-response analysis. `nls` is short for non-linear least squares. The function is included in the base installation of R.

The functions mentioned above are used in a similar way as `lm()`: a model is fitted with the function in question, and the model object subsequently examined with respect to model

validation, estimation, computation of confidence limits, hypothesis tests, prediction, etc. with functions `summary()`, `confint()`, `drop1()`, `predict()`, `emmeans()`, `pairs()` as indicated above.

End of presentation.