Some R

Nomdo Jansonius – January 30, 2026

Install R in Ubuntu linux

nomdo@nomdo-desktop:~$ `sudo apt-get install r-base`

For additional packages, run R as superuser:

nomdo@nomdo-desktop:~$ `sudo R`

and install the concerning package from within R (for example, the epiDisplay package):

```
install.packages("epiDisplay")
```

The following packages are needed for the topics covered in this manual: `epicalc (R version 3) or epiDisplay (R version 4+), gmodels, lme4, quantreg, Hmisc, MatchIt, lattice, lmerTest, ordinal, betareg, cluster, clusrank (R version 4+), factoextra, MASS, readxl.`

Note: you have to install packages only once; in a script they are activated via `library()`.

Running R

The most convenient (appropriate) way to run R is by using scripts. A script is a text file (foo.r) that contains a series of commands. To run a script in Linux, open a terminal, navigate to the directory where the script and datafile are located, and execute the script by typing (no need to start R first!):

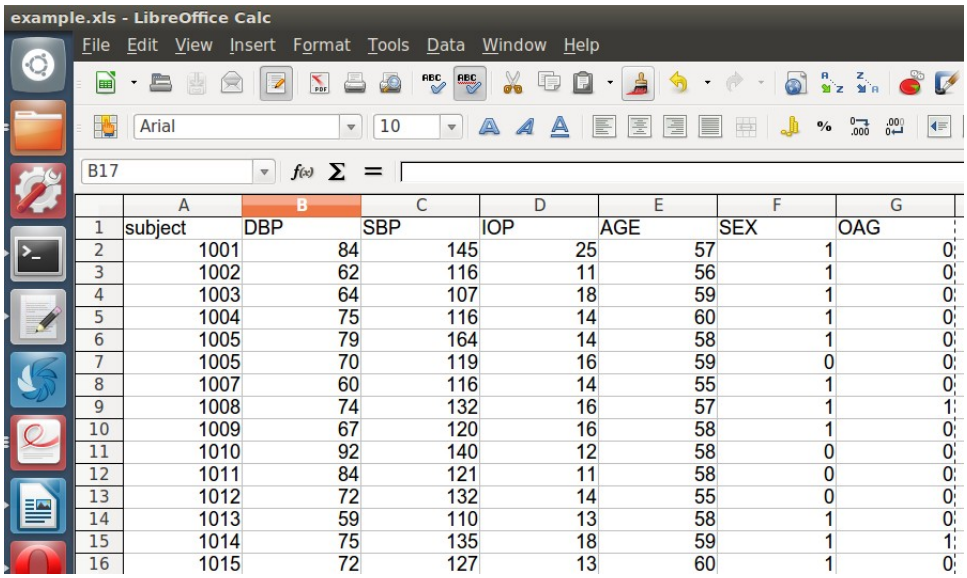
nomdo@nomdo-desktop:~$ `Rscript foo.r [> Rresults.txt]`

Output comes by default to the screen (you may add > Rresults.txt); graphics is stored in Rplots.pdf.

To temporarily deactivate parts of a script, put # for each line or, for longer parts, put the concerning part in between the {} in: `if (FALSE) { }`.

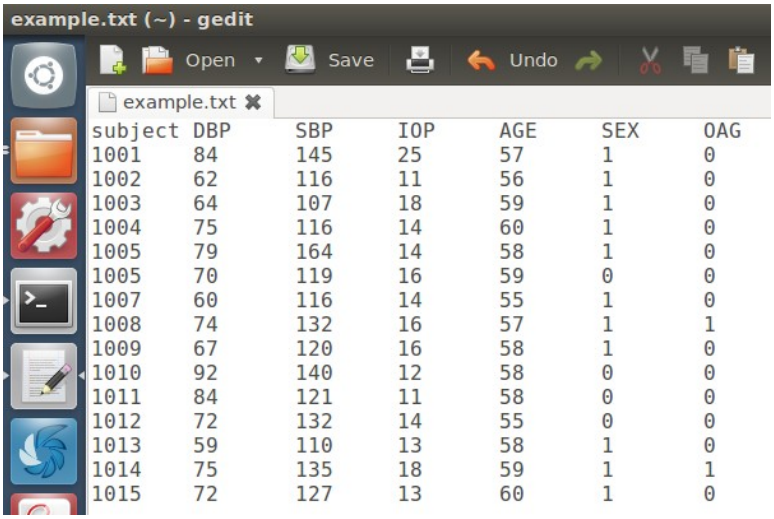
Datafile preparation

Datafiles can be prepared in any spreadsheet program:



	A	B	C	D	E	F	G
1	subject	DBP	SBP	IOP	AGE	SEX	OAG
2	1001	84	145	25	57	1	0
3	1002	62	116	11	56	1	0
4	1003	64	107	18	59	1	0
5	1004	75	116	14	60	1	0
6	1005	79	164	14	58	1	0
7	1005	70	119	16	59	0	0
8	1007	60	116	14	55	1	0
9	1008	74	132	16	57	1	1
10	1009	67	120	16	58	1	0
11	1010	92	140	12	58	0	0
12	1011	84	121	11	58	0	0
13	1012	72	132	14	55	0	0
14	1013	59	110	13	58	1	0
15	1014	75	135	18	59	1	1
16	1015	72	127	13	60	1	0

R is able to import data directly from an xls(x) file (see below), but it uses by default a plain text file for import. For making a plain text file, copy paste the concerning part of the spreadsheet – including the variable names in the first row – into a text editor:




```
subject DBP     SBP     IOP     AGE     SEX     OAG
1001    84      145     25      57      1       0
1002    62      116     11      56      1       0
1003    64      107     18      59      1       0
1004    75      116     14      60      1       0
1005    79      164     14      58      1       0
1005    70      119     16      59      0       0
1007    60      116     14      55      1       0
1008    74      132     16      57      1       1
1009    67      120     16      58      1       0
1010    92      140     12      58      0       0
1011    84      121     11      58      0       0
1012    72      132     14      55      0       0
1013    59      110     13      58      1       0
1014    75      135     18      59      1       1
1015    72      127     13      60      1       0
```

Warning: R uses decimal point (default in English), no comma (as is default in some other languages, including Dutch)!

Missing values

In plain text files, empty cells have to be filled with NA. Highlight the concerning part of the spreadsheet => edit => find & replace => leave the "search for" field empty (or enter 9999 if missing values have been coded as 9999, etc) and enter NA in the "replace with" field => replace all. If data are imported directly from an xls file, empty cells should be filled with a number (e.g., 9999).

If there are missing values, R performs, by default, a complete case analysis for multivariable analysis. In case of a limited number of missing values that scatter around the independent variables, this might be appropriate. If the missing values are mainly concentrated in one or a few independent variables, simply omitting these variables could be a better approach. A third approach is imputation (see below).

Importing data into R

The first command cleans the workspace. The second command imports data from a txt datafile. The last command makes the imported data available within R:

```
rm(list=ls())
mydata <- read.table("example.txt",header=T)          # for txt datafile
attach(mydata)                                        # see also detach/attach below
```

In case of data in an xls or xlsx file, the second line should be replaced by two lines:

```
library("readxl")                                     # for xls or xlsx datafile
mydata <- read_excel("example.xls",na="9999")         # idem
```

The data are now available and the variables are accessible by name. The `header=T(RUE)` statement indicates the presence of the variable names in the first row of the datafile (not needed for the xls file import option). Don't use – (dash) in either variable names or datafile names!

To view the first six rows of the datafile and the variable names:

```
head(mydata)
```

To view some basic descriptive statistics:

```
summary(mydata)
```

Imputation

Imputation of missing values could result in less bias compared to the complete case analysis approach. After all, missing values are not always missing completely at random (MCAR). The probability of having missing values for a variable may be associated with one or more other variables (for example, measuring IOP is more difficult in small children): missing at random (MAR). Also, the probability of having missing values may depend on the variable itself (for example, IOP values outside the range of the tonometer device): missing not at random (MNAR). In case of MCAR, mean imputation is the most straightforward approach:

```
library(Hmisc)
IOPcomplete <- impute(mydata$IOP,mean)          # IOPcomplete is a new variable
```

Imputation of primary variables should be avoided; imputation may be useful to deal with missing data in confounding factors or covariates.

Combining, recoding, or creating new variables

continuous variables

```
C <- A*B
C <- A+B
baselineMD <- pmin(MD2,MD3)
```

logical (dichotomous) variables

```
C <- A&B      AND
C <- A|B      OR
C <- A&!B     A AND NOT B
```

dichotomous variable from a continuous variable

```
OHT <- ifelse(IOP > 20, c(1), c(0))
progression <- ifelse(baselineMD > pmax(MD5,MD6), c(1), c(0))
```

tertiles, etc. from a continuous variable

```
IOPtert <- cut(IOP,breaks=quantile(IOP,c(0,0.33,0.67,1)),
                       labels=c('low','medium','high'),include.lowest=T)
```

Subsets

To make - within R - a complete case dataset (this removes all rows with one or more NA):

```
completecasedataset <- na.omit(mydata)
detach(mydata)
attach(completecasedataset)
```

Subset of entire dataset, *e.g.*, only the cases (subjects with OAG=1):

```
cases <- subset(mydata,OAG==1)
```

Only the first *n* rows of the entire dataset

```
partofdataset <- head(mydata,n)
```
or
```
partofdataset <- mydata[1:n, ]
```

Only young or middle-age participants:

```
jong <- subset(mydata,AGE<40)
```

```
middelbaar <- subset(mydata,AGE<65 & AGE>40)
```

Logical operators that can be used: `==` `!=` `>` `<` `<=` `>=` `&` `|`

Detach/attach

If one or more subsets have been created, variable names become ambiguous. To avoid ambiguity, use `detach/attach` (see example above). Alternatively, do not use `attach` at all but declare the source of the variables explicitly. For individual variables, put `(sub)setname$` before the variable name:

```
mydata$IOP
jong$IOP
```

For models, the (sub)dataset may be declared within the model syntax by using the `data=` switch:

```
model <- glm(OAG ~ IOP+AGE+SEX,family=binomial,data=jong)
```

Models will be covered in detail below.

Counting and identification

Number of rows (subjects) in the datafile:

```
length(subject)
```
[or any other variable name available in the datafile]

Number of subjects with data on IOP available:

```
length(IOP[!is.na(IOP)])
```

Number of subjects with IOP > 20:

```
length(IOP[IOP>20])
```

Identification (subject ID numbers) of all male subjects with IOP > 20:

```
subject[IOP>20 & SEX==0]
```

Note: () for functions and [] for variables.

Displaying categorical variables

```
ftable(SEX)
```

```
library(gmodels)
CrossTable(SEX,OAG)
```

Testing normality

```
qqnorm(IOP)
qqline(IOP,lty=2)
```

or:

```
shapiro.test(IOP)
```

Parametric descriptive statistics

```
mean(IOP)
sd(IOP)
```

Mean and standard deviation cannot be calculated if there are one or more missing values. In case of missing values:

```
mean(IOP[!is.na(IOP)])
```

Nonparametric descriptive statistics

Median (P50):

```
median(IOP)
```

Default: min, P25, median, P75, and max:

```
quantile(IOP)
```

Any other percentile (for example: min, P2.5, median, P97.5, and max):

```
quantile(IOP,c(0,.025,.5,.975,1))
```

Missing values are not allowed. In case of missing values:

```
quantile(IOP[!is.na(IOP)],c(0,.025,.5,.975,1))
```

Correlation analysis

```
cor(IOP,AGE)                          # Pearson correlation coefficient
cor(IOP,AGE,method='spearman')        # Spearman correlation coeffcient
```

To also get a p-value: replace `cor` by `cor.test`

Chi-square test and McNemar test

```
table(SEX,OAG)
chisq.test(table(SEX,OAG))

table(TEST1, TEST2)
mcnemar.test(TEST1, TEST2, correct = TRUE)
```

Creating a contingency table

In the example above, the table was created from a datafile. Contingency tables can also be created directly. To create a matrix with 2 colums and 3 rows, use either `cbind` or `rbind`:

```
x <- cbind(c(16,6,5),c(11,12,5))
```

or:

```
x <- rbind(c(16,11),c(6,12),c(5,5))
```

Next, specify the column names and row names of matrix x:

```
colnames(x) <- c('col1','col2')
rownames(x) <- c('row1','row2','row3')
```

Assign to table:

```
tabel <- as.table(x)
```

Finally, display and calculate chi-square:

```
tabel
chisq.test(tabel)
```
or `chisq.test(x)`

F-test, t-test, and Wilcoxon test

An F-test, t-test, or Wilcoxon rank sum test (Mann–Whitney U test) for independent samples, comparing variable IOP between cases (OAG=1) and controls (OAG=0):

```
var.test(IOP[OAG==1],IOP[OAG==0])          # for choosing between t-test and Welch test
t.test(IOP[OAG==1],IOP[OAG==0],var.equal=T)    # for Welch test: var.equal=F
wilcox.test(IOP[OAG==1],IOP[OAG==0])
```

and the corresponding tests for paired samples (paired t-test, Wilcoxon signed rank [sum] test), comparing SBP and DBP:

```
t.test(SBP,DBP,paired=T)
wilcox.test(SBP,DBP,paired=T)
```

To test if a single sample (for example, IOP) differs from zero:

```
t.test(IOP)
wilcox.test(IOP)
```

Adjustment for multiple comparisons

```
p = c(0.001,0.10,0.049,0.051)                  # vector containing multiple p-values
p.adjust(p, method = "fdr", n = length(p))     # some options: fdr, bonferroni
```

Note: In case of a single test, a p-value depicts the probability of being wrong if you claimed an effect, i.e., the probability of making a type I error (rejecting an actually true null hypothesis). A Bonferroni-corrected p-value refers to the probability of making at least one type I error in case of a series of tests. More commonly, however, not a Bonferroni-corrected p-value is given but an adapted cut-off value for significance: 0.05/n, with n the number of tests). An FDR-corrected p-value (also called q-value) gives the proportion of false-positive 'discoveries' among all 'discoveries'. For example, a q-value of 0.05 means that we should expect 5% of all 'discoveries' with a q-value less than 0.05 to be false positives.

When to adjust? Much ado about that. Ideally, over an entire scientific career… Common occasions are in post-hoc comparisons after a significant one-way or two-way ANOVA or their non-parametric equivalents (Kruskal-Wallis and Friedman's test), and when multiple outcomes are studied in parallel.

Wilcoxon tests for clustered data

This is a recently developed extension of the Wilcoxon tests. It seems useful for vision research, where observations from different eyes in the same subject cannot be considered independent observations. There are two variants, one for independent samples and one for paired samples, being the counterparts of the rank sum test (Mann-Whitney U test) and signed rank (sum) test, respectively.

The datafile organisation for the `clusWilcox.test` should look like:

x	y	subject	groep	eye
3.0	6.5	1	1	1
4.0	7.8	1	1	2
7.0	9.9	2	1	1
6.0	8.7	2	1	2
4.1	6.3	3	1	1
3.1	5.9	3	1	2
...	...	.	.	.
7.1	9.7	100	2	1
6.1	9.8	100	2	2

where variable *y* is only needed in case of paired samples, variable 'groep' only in case of independent samples, and variable 'eye' is not needed in the syntax of either variant, but helps to understand the datafile structure.

For <u>independent samples</u> (variable *x* measured in two different groups):

```
library(clusrank)
clusWilcox.test(x,group=groep,cluster=subject)
```

and for <u>paired samples</u> (*x* and *y*):

```
library(clusrank)
clusWilcox.test(x,y,paired=T,cluster=subject)
```

which equals:

```
z <- x-y
clusWilcox.test(z,paired=T,cluster=subject)
```

Some basic (and not so basic) plotting

Histogram

```
  hist(IOP)
```

controlling range and binwidth

```
  hist(IOP, breaks=seq(from=0, to=30, by=1))
```

Scatterplot

```
  plot(IOP~AGE)
```

adding jitter to scatterplot

```
  plot(jitter(IOP,1)~jitter(AGE,1))    # 1 can be changed to modify the amount of jitter
```

Boxplot

with one variable describing the boxes

```
  boxplot(IOP~SEX)
```

with more than one variable describing the boxes

```
  boxplot(IOPpre,IOPpost)
```

Jitterplot

```
  library(lattice)
```

basic

```
  stripplot(IOP~SEX,jitter.data=T)
```

black instead of blue datapoints

```
  stripplot(IOP~SEX,jitter.data=T,col="black")
```

nicer x-axis

```
  SEXF <- factor(SEX)
  stripplot(IOP~SEXF,jitter.data=T)
```

meaningful names for coded categorical variables

```
  SEX[SEX==0] <- "MALE"
  SEX[SEX==1] <- "FEMALE"
  stripplot(IOP~SEX,jitter.data=T)
```

Update order of categorical variables (default is alphabetical)

```
  SEX_ordered <- factor(SEX, c("MALE","FEMALE"))
```

Labels

```
  plot(IOPpost~IOPpre, xlab="IOP pre-op (mmHg)", ylab="IOP post-op (mmHg)")
  boxplot(IOPpre, IOPpost, ylab="IOP (mmHg)", names=c("pre-op","post-op"))
```

Range of axes

```
  plot(IOP~AGE, xlim=c(0,100), ylim=c(0,40))
```

Modify tics

```
plot(... , xaxt="n", …)                    # modify tics x-axis
axis(1,at=c(x1,x2,x3,x4,x5))

plot(... , yaxt="n", …)                    # modify tics y-axis
axis(2,at=c(y1,y2,y3,y4,y5))
```

Font size of labels and numbers

```
  plot(... ,cex.lab=1.25, cex.axis=1.25)   # increase font size by 25%
```

Lines

```
plot(IOP~AGE)
abline(h=20)              # horizontal line; continuous
abline(h=20, lty=2)       # horizontal line; dashed
abline(0,1)               # line with slope 1 and intercept 0, that is, y=x
```

Title at top of figure

```
plot(..., main="Title")   # no title: main=""
```

Output to file

```
postscript("output.eps")
... plotting commands ...
dev.off()
```

Note: `postscript("output.eps")` may be replaced by `pdf("output.pdf")` or by `tiff("output.tif",compression="lzw",width=6,height=6,units='in',res=300)`. For tif, font size is independent of width and height, that is, lower width and height values (e.g., 4) yield relatively larger numbers and labels.

Some important notes for multivariable models – readme.1st!

1) If a categorical variable has more than two categories (e.g., no, low, and high myopia coded as 0, 1, and 2, respectively), then the variable must be declared as being categorical:

```
myopiaF <- factor(myopia)
```

2) An interaction term between two independent variables (for example, A and B) can be entered in the model by replacing A+B by either A*B or A+B+A:B

```
model <- glm(OAG ~ IOP+AGE*SEX,family=binomial,data=mydata)
```

Interpretation is not easy! Please see: https://www.theanalysisfactor.com/interpret-main-effects-interaction/ and https://www.theanalysisfactor.com/interactions-main-effects-not-significant/

3) There are two main aims to build a model: 1. confirmatory modeling (hypothesis driven) and 2. exploratory/predictive modeling (hypothesis generating/free).

Confirmatory modeling (hypothesis driven)

Here, the aim is to study the effect of one independent variable. Modeling is used to adjust for (potential) confounding factors, in order to reduce bias or increase precision. You may simply add the potential confounding factors to the model, irrespective of their p-values. It is common to report i. the crude effect size of the independent variable, ii. the effect size adjusted for age and gender, and iii. the effect size adjusted for age, gender, and all other potential confounding factors for which you adjusted.

Predictive/exploratory modeling (hypothesis free/generating)

Here, a possible approach is the following. Start with a saturated model, that is, a model with all independent variables (predictors) included. The least significant variable is then removed and the models with and without the least significant variable are compared using the Akaike information criterion (AIC). If the model without the concerning variable is the better fit (has a lower AIC), then the same process is repeated for the next least significant variable, and so on. Practically, this equals removing the variables one by one until all remaining p-values are lower than 0.157.

Note: in confirmatory modeling, missing data should be imputed; in predictive and exploratory modeling, a complete case approach is considered best.

Multiple linear regression

```
pairs(mydata)
cor(mydata)  # in case of missing data: cor(na.omit(mydata))

model <- lm(IOP ~ AGE+SEX)
summary(model)
plot(model)            # to check, among others, if the residuals are normally distributed
```

Logistic regression

```
model <- glm(OAG ~ IOP+AGE+SEX,family=binomial,data=mydata)
```

Here, the dependent variable is the variable before the ~ and binomial points to logistic regression (model is an arbitrarily chosen name). To display the results on the screen:

```
summary(model)
```

Or, to get a more familiar logistic regression presentation with ORs:

```
library(epiDisplay)
logistic.display(model)
```

Cox regression

For Cox regression, a variable is needed that describes the follow-up duration, that is, the time from baseline to either event or censoring. Below, this variable is named FU; dependent variable is iOAG.

```
library(epiDisplay)
model <- coxph(Surv(FU,iOAG) ~ IOP+AGE+SEX,data=mydata)
summary(model)
```

Regression with count data (non-negative integer)

```
model <- glm(count ~ IOP+AGE+SEX,family=poisson,data=mydata)
summary(model)
```

Regression with positive continuous data (real number > 0)

```
model <- glm(R+ ~ IOP+AGE+SEX,family=Gamma,data=mydata)
summary(model)
```

Regression with proportion data (real number [0,1])

A proportion ranges from 0 to 1, including 0 and 1; beta-regression requires an open interval (0,1) for the dependent variable. If this is not the case, first transform the proportion:

```
proportionB <- (proportion+0.001)/1.002
```

Next:

```
library(betareg)
model <- betareg(proportionB ~ IOP+AGE+SEX)
summary(model)
```

Regression with Likert scale data (ordinal factor with 4-10 items)

```
library(ordinal)
ratingF <- factor(rating)
model <- clm(ratingF ~ IOP+AGE+SEX,data=mydata)
summary(model)
```

ANOVA

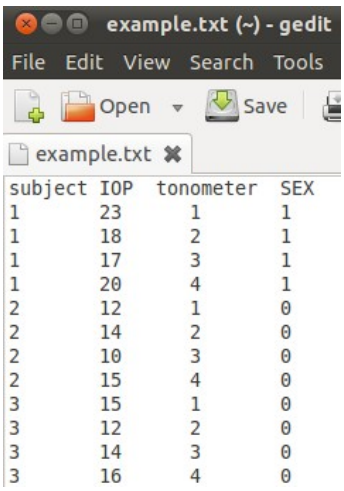
If the factors (categorical explanatory variables) are coded as numbers, they have to be declared as being categorical first:

```
tonometerF <- factor(tonometer)
SEXF <- factor(SEX)
model <- aov(IOP ~ tonometerF*SEXF)
summary(model)
```

Post-hoc test

```
TukeyHSD(model)
```

Repeated measures



subject	IOP	tonometer	SEX
1	23	1	1
1	18	2	1
1	17	3	1
1	20	4	1
2	12	1	0
2	14	2	0
2	10	3	0
2	15	4	0
3	15	1	0
3	12	2	0
3	14	3	0
3	16	4	0

If there are repeated measures (the same subject is present in more than one row of the datafile), then subject must be added as a variable (the datafile has to contain a column with subject number) and this variable has to be declared as being categorical:

```
subjectF <- factor(subject)
```

And subsequently:

```
model <- aov(IOP ~ tonometerF*SEXF + Error(subjectF/tonometerF))
summary(model)
```

Here, tonometerF is a within-subject and SEXF a between-subject variable. Generally:

```
model <- aov(DV ~ b1*b2)
model <- aov(DV ~ w1*w2 + Error(subject/(w1*w2)))
model <- aov(DV ~ b1*w1 + Error(subject/w1))
model <- aov(DV ~ b1*b2*w1 + Error(subject/w1))
model <- aov(DV ~ b1*w1*w2 + Error(subject/(w1*w2)))
```

where DV is the dependent variable, b1 and b2 between-subject variables, and w1 and w2 within-subject variables.

The dependent variable in ANOVA must be continuous; independent variables may be either categorical (*e.g.*, SEX) or continuous (*e.g.*, age):

```
SEXF <- factor(SEX)
model <- aov(IOP ~ age*SEXF)
```

Note: Repeated measures ANOVA using aov is a Type I analysis and cannot handle missing values or unbalanced (non-orthogonal) data. So, if you want to keep things simple, be sure that all conditions (all possible combinations of within-subject variables) are measured in all subjects and that all groups (all possible combinations of between-subject variables) have equal size. If the order of the between-subject variables in the model matters, significant imbalance exists. If so, a Type II or Type III analysis could be performed. However, a better approach to unbalanced data is using lmer (see below).

For two-way ANOVA used to analyze repeated measures in a single group of subjects:

```
model <- aov(DV ~ w + Error(subject/w))
```

The TukeyHSD test will not run on this model. In fact, it will not run on any model with an Error term (that is, with repeated measures). A solution is to use pairwise paired t-tests with adjusted p-values:

```
with(mydata,pairwise.t.test(DV,w,p.adjust.method="bonferroni",paired=T))
```

where mydata is the name of the dataset, DV the dependent variable, and w the within-subject variable for which you want to do the post-hoc analysis. In Bonferroni, the p-values are multiplied by the number of comparisons. To calculate the corresponding mean values of DV for the various categories:

```
mean(DV[w==1])
mean(DV[w==2])
mean(DV[w==3])
```

Linear mixed effects models

An approach that allows for handling repeated measures in a much more flexible way than ANOVA.

```
library(lme4)
library(lmerTest)                    # optional; to get p-values when using summary
model <- lmer(DV ~ f1+f2+(1|r1)+(1|r2),REML=FALSE)
summary(model)
plot(model)                          # to get an impression of the distribution of the residuals
```

Here, f1 and f2 are fixed effects and r1 and r2 random effects. Fixed effects are independent variables that you want to study (e.g., yes/no treatment, case/control), covariates, or confounders (age, gender, etc). Random effects are variables for which you have to adjust in order to avoid the spurious effects related to dependent observations. If lmer is used to evaluate repeated measures, then subject has to be added as a random effect. A lmer model needs at least one random effect (after all, it's a mixed effects model - if not, use lm). Don't forget to declare categorical variables (with more than two categories) as being categorical.

By default, lmer uses REML. If you are comparing models with different fixed effects via hypothesis tests or information-theoretic criteria such as AIC, then REML should be set to FALSE.

Random effects can be nested or crossed. If you collected several observations from both eyes in one or more subjects in your study group, then the random effects are nested: eye 1 of subject 1 is another eye than eye 1 of subject 2. The correct way of modelling nested random effects:

```
model <- lmer(DV ~ f1+f2+(1|subject/eye),REML=FALSE)
```

Note: if the only reason to use mixed effects models was the inclusion of both eyes in one or more subjects, but only one measurement was performed in each eye, then you only need `(1|subject)`; adding `/eye` would be redundant and results in an error message.

If you used, for example, different (types of) tonometers to assess intraocular pressure, but tonometer 1 used in subject 1 was the same tonometer as tonometer 1 used in subject 2, then subject and tonometer are crossed random effects. The correct way of modelling crossed random effects:

```
model <- lmer(DV ~ f1+f2+(1|subject)+(1|tonometer),REML=FALSE)
```

Nested and crossed random effects may coexist. For a multicenter study where different tonometers were used and individual participating centers used more than one tonometer:

```
model <- lmer(DV ~ f1+f2+(1|center/subject/eye)+(1/tonometer),REML=FALSE)
```

Note: if different tonometers were used but each center used only one tonometer, then tonometer should not be added as a random effect: it's variability is already accounted for in 'center'.

A variable may be both a fixed effect and a random effect, but the interpretation differs:

```
modelA <- lmer(IOP ~ treatment+(1|subject)+(1|tonometer),REML=FALSE)
modelB <- lmer(IOP ~ tonometer+(1|subject),REML=FALSE)
```

In modelA, the effect of treatment is studied, and because different tonometers were used, tonometer is included as a random effect. In modelB, differences between different tonometers are studied.

For confirmatory modeling, to assess the contribution/significance of, for example, f1, make a second model without f1 and compare it to the original model:

```
model.without.f1 <- lmer(DV ~ f2+(1|r1)+(1|r2),REML=FALSE)
anova(model,model.without.f1)
```

For exploratory modeling, a simplified option is to define all possible models (with at least one random effect) and compare them simultaneously. The 'best' model is the model with the lowest AIC:

```
m0 <- lmer(DV ~ f1*f2+(1|r1)+(1|r2),data=mydata,REML=FALSE)
m1 <- lmer(DV ~ f1+f2+(1|r1)+(1|r2),data=mydata,REML=FALSE)
m2 <- lmer(DV ~ f1+f2+(1|r1),data=mydata,REML=FALSE)
m3 <- lmer(DV ~ f1+f2+(1|r2),data=mydata,REML=FALSE)
m4 <- lmer(DV ~ f1+(1|r1)+(1|r2),data=mydata,REML=FALSE)
m5 <- lmer(DV ~ f2+(1|r1)+(1|r2),data=mydata,REML=FALSE)
m6 <- lmer(DV ~ f1+(1|r1),data=mydata,REML=FALSE)
m7 <- lmer(DV ~ f1+(1|r2),data=mydata,REML=FALSE)
m8 <- lmer(DV ~ f2+(1|r1),data=mydata,REML=FALSE)
m9 <- lmer(DV ~ f2+(1|r2),data=mydata,REML=FALSE)

anova(m0,m1,m2,m3,m4,m5,m6,m7,m8,m9)
summary(mx)          # with mx the model with the lowest AIC
confint(mx)          # to get confidence intervals for the estimates
```

Linear mixed effects models to study change over time

In case of repeated measures (DV has been measured several times in each subject) aiming to assess (a linear) change over time, we add a slope to the mixed effects model:

```
m0 <- lmer(DV ~ time+(1|subject))              # effect of time on DV ('random
                                                 intercept model')
m1 <- lmer(DV ~ time+(time|subject))           # differences in individual slopes
                                                 ('random slope model')
m2 <- lmer(DV ~ age+time+(time|subject))       # effect of age on DV
m3 <- lmer(DV ~ age*time+(time|subject))       # effect of age on individual slopes
m4 <- lmer(DV ~ gender+age*time+(time|subject))
m5 <- lmer(DV ~ gender*time+age*time+(time|subject))
```

Generalized linear mixed effects models

If the dependent variable is not a continuous variable that is not bounded, lmer cannot be used. Instead, use **g**lmer together with the appropriate family (poisson, Gamma, binomial). For example:

```
library(lme4)
model <- glmer(count ~ f1+f2+(1|r1)+(1|r2),family=poisson)
```

Note: the REML switch should not be used together with glmer.

Linear discrimimant analysis (LDA)

```
library(MASS)
model <- lda(group ~ v1+v2+ ... +vm)
model
plot(model)
```

The input datafile should contain columns with independent variables (v_1, v_2, …, v_m) and a column with the ID of the group to which a subject belongs (group; categorical dependent variable). With two groups, the output will be the coefficients of the discriminant function. This function can be used to build, e.g., an ROC curve. With k groups with $k > 2$, the output will be $k - 1$ series of coefficients (data reduction from m to $k - 1$ variables). Importantly, m must be smaller than the group size of the smallest group. The independent variables v_i should be continuous and normally distributed. Normality seems not critical, though, and possibly even dichotomous variables may be used. If the assumptions are violated, you may value the results of your analysis concerning two groups by evaluating the AUC.

Principle component analysis (PCA)

Related to LDA, but now the group to which a subject belongs is unknown, or at least not used in the analysis (PCA = unsupervised learning; LDA = supervised learning). LDA is primarily a tool for separating groups; PCA is primarily a tool for data reduction (it reduces the dimensionality of a dataset while maintaining most of the information).

```
inputdata <- mydata[, 2:5]
model <- prcomp(inputdata, center=TRUE, scale=TRUE)
summary(model)
print(model)
plot(model)     #scree plot
biplot(model)  #loading plot
```

Now, the data are imported as a matrix (inputdata) that is a subset of the attached dataset (mydata; in this example the columns 2 to 5 – especially <u>not</u> including any group ID). Switch center = TRUE refers to subtracting the mean of each variable from each variable; switch scale = TRUE refers to dividing by the standard deviation. This normalization should be the default approach unless all variables have the same unit and differences in magnitude have a meaning.

A variable should be either a normally distributed continuous variable or a (numerically coded – do not make a factor out of it!) dichotomous (binary) variable. Applying PCA to Likert scale data is controversial. Making a Likert scale binary, by using a well-reasoned cut-off, is one possible solution.

The print command shows the standard deviations (the square root of the eigenvalues) of the principal components and the coefficients of the linear combinations of the variables describing the components.

The principal components are orthonormal vectors; together the span an *m*-dimensional space, with *m* the number of variables. The direction of the first principal component coincides with the direction of the largest variability in the datapoints (i.e., the direction of a line for which the sum of the squared perpendicular distances between the line and the datapoints has been minimized).

If the variables are normalized, the sum of the eigenvalues equals the number of independent variables, *m*, and the variance explained by a component equals the corresponding eigenvalue divided by *m*.

The scree plot presents the ranked eigenvalues of all principal components, which facilitates the inclusion/exclusion. As a rule of thumb, components with an eigenvalue below 1 should be excluded: they explain less than an original variable. The final number of included components depends on the desired simplicity, and on the required accuracy (overall explained variance).

The loading plot shows how the original variables contribute to the first two components. This plot can be used to interpret the meaning of these components.

To make a nice scatterplot of the data for PC2 versus PC1:

```
plot(model$x[,1:2], pch=20)
```

where *x* is a matrix containing the data after rotation (transformation from the original variables to the principal components). If more than one group was present in the dataset, you could use different colours for datapoints belonging to different groups:

```
plot(model$x[,1:2], pch=20, col=mydata$group)
```

where group is a variable containing the group ID, which can be used to code the color if positive integers are used (1=black, 2=red, 3=green, 4=blue, 5=cyaan, 6=magenta).

Cluster analysis

Related to both LDA and PCA. Like in PCA, the group to which a subject (row) belongs is unknown, or at least not used in the analysis (unsupervised learning); like LDA, cluster analysis is primarily a tool for separating groups.

```
library(factoextra)
library(cluster)

mydata <- iris[,1:4]
```

Dataset 'iris' is a built-in dataset in R; columns 1 to 4 contain the independent variables; the fifth (removed) column of iris contains the species (categorical dependent variable).

Remove rows with missing values, if any, and scale each variable to have mean 0 and SD 1:

```
mydata_complete <- na.omit(mydata)
mydata_normalised <- scale(mydata_complete)
```

Create a plot of the number of clusters versus the 'total within sum of squares'; optimal k is where the curve starts to flatten:

```
fviz_nbclust(mydata_normalised, kmeans, method = "wss")
```

Perform k-means clustering with k = 3 clusters:

```
set.seed(1)
km <- kmeans(mydata_normalised, centers = 3, nstart = 25)
```

Add the cluster assigment to the original data:

```
mydata_with_cluster <- cbind(mydata_complete, cluster = km$cluster)
```

Find means of each cluster and visualize the clusters on a scatterplot that displays the first two principal components:

```
aggregate(mydata_complete, by=list(cluster=km$cluster), mean)
fviz_cluster(km, data = mydata_normalised)
```

Cluster analysis needs continuous variables. If a dataset contains one or more binary variables, do a PCA first and then perform cluster analysis on the PCs.

Quantile regression

```
  library(quantreg)
  model <- rq(IOP ~ AGE, tau=0.5)      # tau: percentile (default: median)
  summary(model)                       # with confidence intervals
  summary(model,se="nid")              # with p-values
```

Bland-Altman plot and analysis

Code for making a Bland-Altman plot and calculating the corresponding bias and limits of agreement. This also illustrates the use of a function in R. Just copy the code into your script (no modifications needed) and call it with the appropriate variable names:

```
blandaltman <- function(v1,v2) {
  # basic calculations
  means <- (v1+v2)/2
  diffs <- v1-v2
  mdiff <- mean(diffs)
  sddiff <- sd(diffs)
  # range for y-axis
  yh <- mdiff+3*sddiff
  yl <- mdiff-3*sddiff
  # Bland-Altman plot
  plot(means,diffs,xlab="Average values",ylab="Differences",ylim=c(yl,yh))
  abline(h = mdiff)
  abline(h = mdiff+1.96*sddiff, lty=2)
  abline(h = mdiff-1.96*sddiff, lty=2)
  # calculation of bias and limits of agreement
  bias = mdiff
  LoA = 1.96*sddiff
  print(bias)
  print(LoA)
}

blandaltman(GAT,NCT)
```

Propensity score and propensity score matching

Effect of exposure (or treatment) is ideally studied with an RCT but often an observational study is the only feasible option. In an observational study, those who are exposed may differ from those who are not exposed. To minimize the influence of confounding, multivariable analysis can be used, but only if the number of subjects (*n*) is large and the number of potential confounders limited (less than *n*/10). If there are too many potentially relevant confounders, the concept of propensity score may be used. For calculating the propensity score, the confounders are entered in a logistic regression model to predict the *exposure* of interest, *without* including the outcome:

```
ps <- glm(STATINS ~ IOP+AGE+SEX+DBP+SBP+MYOPIA, family=binomial)
summary(ps)
```

As a result, the confounders are collapsed into a "single" variable, the propensity score, representing the probability (propensity) of being exposed. The propensity score can be added to the datafile:

```
mydata$psvalue <- predict(ps, type="response")
write.csv(mydata, file = "mydatawithps.csv", row.names=FALSE)
```

and subsequently used in a multivariable model relating the outcome to the exposure, as if it were the only confounder:

```
outcome ~ exposure + psvalue
```

Alternatively, you may match (1:1 or 1:*m* with ratio *m* a positive integer) those who were exposed to those who were not, using the propensity score, and save a matched dataset for further analysis:

```
library(MatchIt)
m.out = matchit(STATINS ~ IOP+AGE+SEX+DBP+SBP+MYOPIA, data=mydata,
                          method="nearest", discard="both", ratio=1)
summary(m.out)
plot(m.out, type="jitter")
matcheddata <- match.data(m.out)
write.csv(matcheddata, file = "matcheddata.csv", row.names=FALSE)
```

And for subsequent analysis:

```
rm(list=ls())
mydata <- read.csv("matcheddata.csv",header=T)
attach(mydata)
```

Random numbers

Random number from a normal distribution with mean 0 and standard deviation 1:

```
rnorm(1)
```

Random number from a uniform distribution between 0 and 1:

```
runif(1)
```

Three random numbers from a uniform distribution between 50 and 70:

```
runif(3,min=50,max=70)
```

Seven integers between 1 and 100:

```
sample(1:100,7,replace=T)        # replace=F if each number may be chosen only once
```

For Loop

A For Loop can be used for, for example, resampling.

```
n <- 30
resultsvector <- rep(0,n)
for (i in 1:n)
{
  outcome <- rnorm(1)            # arbitrary piece of R script resulting in an outcome value
  resultsvector[i] <- outcome    # outcome value of this run put in resultsvector
}
mean(resultsvector)
sd(resultsvector)
```

Resampling techniques

Bootstrapping

A method to calculate a standard error or confidence interval for any statistic for which there is no easy way to calculate a standard error or confidence interval.

Bootstrapping is based on 'random sampling with replacement'. The successive steps are:

1. draw a sample
2. calculate the statistic of interest
3. repeat this many times
4. the standard deviation of the estimates of the statistic found during the repeats is the aimed standard error of the statistic of interest

Note: a sample must have the same size as the original dataset, but may contain some observations more than once and some observations may be missing, for example: [1,2,3,4,5,6] => [3,5,5,1,1,4].

Note: how many repeats? For estimating a standard error, 30 repeats may be sufficient; however, commonly 1000 repeats or more are used.

k-fold cross validation

A method to estimate how well a model fitted to a specific dataset would perform in an independent dataset, in situations where there is no independent dataset. The successive steps are:

1. split the dataset in k *randomly* sampled subsets ('folds') of (approximately) equal size (for stratified data [case/control; disease stages within the cases], sample proportionally: 'stratified k-fold cross validation')
2. repeat k times:
    i. build the model based on k – 1 subsets (training dataset)
    ii. evaluate the model in the remaining subset (test dataset)
3. mean and standard deviation of the k evaluations represent the generalized performance with standard error of the model

Note: a typical value of k is 10. Leave-one-out cross validation is a special case with k = N where N the size of the entire dataset.

Note: this is sampling without replacement: each subset is once test dataset and belongs k – 1 times to the training dataset).