Some R

Nomdo Jansonius – January 30, 2021 – R version 3.0.2

Install R in Ubuntu

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

For additional packages:

nomdo@nomdo-desktop:~$ `sudo apt-get install r-cran-epicalc r-cran-gmodels r-cran-lme4 r-cran-quantreg r-cran-hmisc r-cran-matchit r-cran-lattice`

(note: package epicalc has been replaced by epidisplay in more recent R versions)

Not all packages are available via `sudo apt-get install r-cran-foo`. If needed, run R as superuser:

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

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

```
install.packages("missForest")
```

Running R

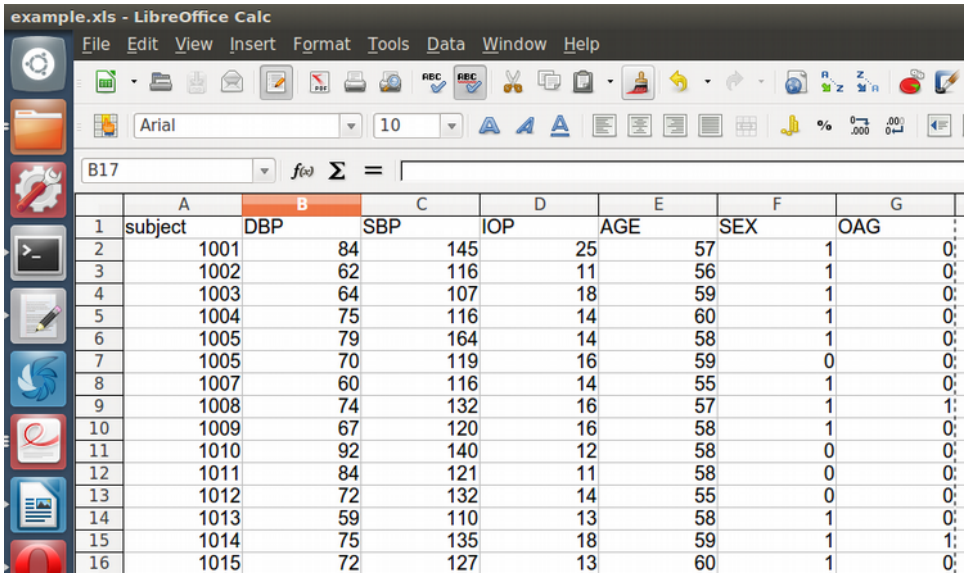
The most convenient (appropriate) way to run R is by using scripts. A script is a text file (foo.r) that contains a series of commands. To run a script in Linux, open a terminal, navigate to the directory where the script and datafile are located, and execute the script by typing (no need to start R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

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

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

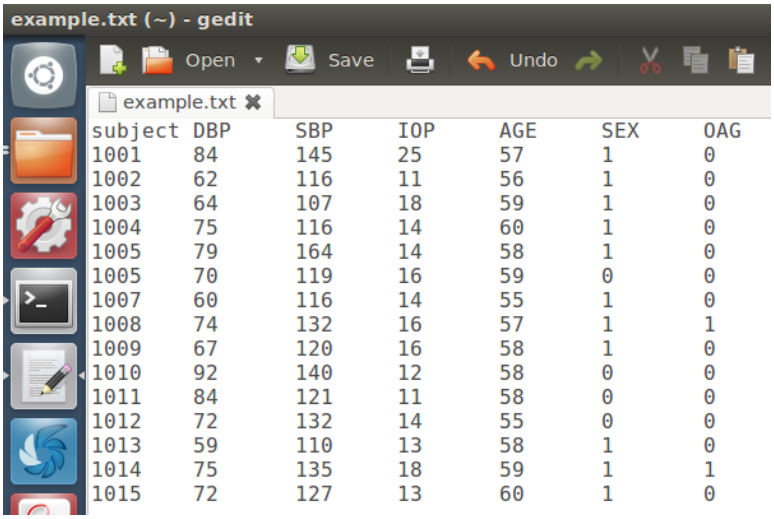
Datafile preparation

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 uses plain text files for import. Hence, copy paste the concerning part of the spreadsheet – including the variable names in the first row - into a text editor:



Warning: R uses decimal point (default in English), no comma (default in Dutch)!

Writing scripts

A script is a text file with a series of R commands. The first command aims to clean the workspace; the second and third command aim to import data from a datafile and to make the data available:

```
rm(list=ls())
mydata <- read.table("example.txt",header=T)
attach(mydata)
```

The data are now available and the variables are accessible by name. The statement "header=T(RUE)" indicates the presence of the variable names in the first row of the datafile.

Now you may view some basic descriptive statistics:

```
summary(mydata)
```

Combining/recoding variables

continuous variables

```
C <- A*B
C <- A+B
```

logical (dichotomous) variables

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

dichotomous variable from a continuous variables

```
OHT <- ifelse(IOP > 20, c(1), c(0))
```

Missing values

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.

By default, R performs - for multivariable analysis - a complete case analysis if missing values are found in a datafile.

In case of a limited number of missing values in a dataset for multivariable analysis, a complete case analysis might be an appropriate approach. Alternatively, if the missing values are mainly concentrated in one or more independent variables, or categories within a variable, simply omitting these variables or categories could be a better option. A third approach is imputation.

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). Finally, for example, repeated measures ANOVA needs a complete, balanced dataset. The imputation methods described below may be used for missing values in continuous variables due to MCAR or MAR.

Mean imputation, per variable (adds a new variable to the original data matrix)

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

Imputation based on all available data (results in a new data matrix)

```
library(missForest)
mydata.imp <- missForest(mydata,variablewise=T)
mydata.imp$OOBerror
attach(mydata.imp$ximp)          # mydata.imp$ximp (not mydata.imp) is a new data matrix
```

Subsets

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

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

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

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

Only young or middle-age participants:

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

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

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

The "attach" is crucial, otherwise the variable names will only refer to the entire dataset (or any previously defined subset), not to the current subset! As soon as one or more subsets have been created, variable names become ambiguous. Now, the source of the variable must be declared explicitly. For individual variables, this can be done by putting `(sub)setname$` before the variable name:

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

For models, the dataset must be declared within the model syntax by using the `data=` switch:

```
jong$OAG <- factor(jong$OAG)
jong$SEX <- factor(jong$SEX)
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 or with IOP > 20:

```
length(IOP[!is.na(IOP)])
length(IOP[IOP>20])
```

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

```
subject[IOP>20]
```

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

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)])
```

Testing normality

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

or:

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

Displaying categorical variables

```
ftable(SEX)

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

Some basic (and not so basic) plotting

Histogram

```
hist(IOP)
```

Scatterplot

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

Boxplot

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

Jitterplot

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

Meaningful names for coded categorical variables

```
SEX[SEX==0] <- "MALE"
SEX[SEX==1] <- "FEMALE"
```

Update order of categorical variables (default is alphabetical)

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

Labels

```
plot(IOP~AGE, xlab="Age (years)", ylab="IOP (mmHg)")
```

Range of axes

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

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
```

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.

Chi-square test

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

F-test, t-test, and Wilcoxon test

An F-test, t-test, or Wilcoxon 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, 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)
```

Important notes for multivariable models – readme.1st!

1) If an (independent) 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:

```
myopia <- 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)
```

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

Logistic regression

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

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

```
summary(model)
```

Or, to get a more familiar logistic regression presentation:

```
library(epicalc)
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(epicalc)
model <- coxph(Surv(FU,iOAG) ~ IOP+AGE+SEX)
summary(model)
```

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, amongst others, if the residuals are normally distributed
```

Regression with proportion data (real number [0,1]) **or count data** (nonnegative integer) **as dependent variable**

```
model <- glm(proportion ~ IOP+AGE+SEX,family=binomial)
summary(model)

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

ANOVA

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

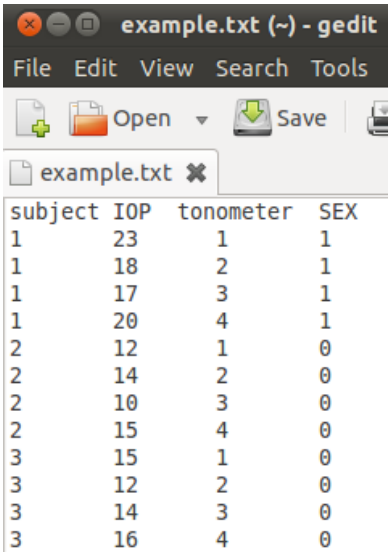
```
tonometer <- factor(tonometer)
SEX <- factor(SEX)

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

Post-hoc test

```
TukeyHSD(model)
```

Repeated measures

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:



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

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

And subsequently:

```
model <- aov(IOP ~ tonometer*SEX + Error(subject/tonometer))
summary(model)
```

Here, tonometer is a within-subject and SEX 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):

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

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 (why wouldn't you?), 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 variables in the model matters, significant imbalance exists. With different group sizes, a Type II analysis could be performed. This can be done using aov by running a series of models in which the position of the between-subject variables is permuted and only the *P* value corresponding to the last between-subject variable is recorded. For example:

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

gives the effect of b2 (and w1 and the interaction terms) after adjustment for b1. The position of the within-subject variables is not relevant nor is the position of the interaction terms. A Type III analysis for unbalanced datasets, preferred by some stats gurus but maligned by others, is possible with R as well but beyond the scope of this tutorial (and my R knowledge).

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])
```

MANOVA

If there is more than one dependent variable, you need a multivariate ANOVA: MANOVA. First, combine the dependent variables into a single variable and subsequently, build and display the model:

```
Y <- cbind(DV1,DV2)
model <- manova(Y ~ b1*b2)
summary(model)
```

If one or more significant independent variables exist, use univariate statistics to see which differ:

```
summary.aov(model)
```

In fact, this is identical to running separate ANOVAs for each dependent variable. If appropriate, the separate ANOVAs may be followed by post-hoc analysis.

Note: you need repeated measures ANOVA if you measure the same thing more than once; you need MANOVA if you measure different things simultaneously.

Linear mixed effects models

```
library(lme4)
model <- lmer(DV ~ f1+f2+(1|r1)+(1|r2),REML=FALSE)
summary(model)
```

Here, f1 and f2 are fixed effects and r1 and r2 random effects. To assess the contribution/significance of f1 in this model, 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)
```

Fixed or random effect? Much ado about this issue... Age, gender, yes/no treatment, and case/control are examples of fixed effects; 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 (it's a mixed model – if not, use lm). Don't forget to declare subject and other categorical variables as being categorical!

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).

Quantile regression

```
  library(quantreg)
  model <- rq(IOP ~ AGE+SEX)
  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)
```