# Faktorielle Versuchspläne in R

Schauen wir uns an, wie man einen faktoriellen Versuchsplan in R auswertet. Wir beginnen mit einem Versuchsplan mit zwei Faktoren.

Ein Arbeitnehmer will untersuchen, ob die Strecke A und der Zeitpunkt B der Abfahrt einen Einfluss auf die Fahrzeit zur Arbeit haben.
Es werden also die folgenden zwei Faktoren betrachtet:

A: Stecke mit den Faktorstufen 1 (-) und 2 (+)
B: Zeitpunkt der Abfahrt mit den Faktorstufen früh (-) und spät (+)

Auf jeder Faktorstufenkombination wurde ein Versuch durchgeführt und die Anzahl der Wörter bestimmt, die aufgeschrieben wurden.

Hier sind die Daten:

```
 A B Fahrzeit
 - -    40
 + -    46
 - +    42
 + +    50
```

Wir geben die Daten ein.

```
> Zeit<-c(40,46,42,50)
> Strecke<-factor(c(-1,1,-1,1))
> Zeitpunkt<-factor(c(-1,-1,1,1))
```

Wenn wir das additive Modell unterstellen, erhalten wir die ANOVA-Tabelle folgendermaßen:

```
> summary(aov(Zeit~Strecke+Zeitpunkt))

            Df Sum Sq Mean Sq F value  Pr(>F)
Strecke      1     49      49      49 0.09033 .
Zeitpunkt    1      9       9       9 0.20483
Residuals    1      1       1
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1
```

Schauen wir uns diese Tabelle genauer an. Df steht für Freiheitsgrade, Sum Sq für Quadratsummen , Mean Sq für mittlere Quadratsummen und F value für den Wert der Teststaistik des F-Tests. Außerdem gibt es noch eine Spalte, die mit Pr(>F) überschrieben ist. Hier findet man die *Überschreitungswahrscheinlichkeit* des F-Tests. Dies ist das kleinste Signifikanzniveau, zu dem man die Nullhypothese ablehnen würde. Führt man den Test also zum Niveau 0.05 durch, so lehnt man die Nullhypothese ab, wenn die Überschreitungswahrscheinlichkeit kleiner als 0.05 ist. Wir sehen, dass der Effekt keines Faktors signifikant ist.

Um die Effekte zu erhalten, muss man den Algorithmus von Yates durchführen. Dies leistet die folgende Funktion:

```
yates<-function(v,alles=TRUE)
 { # Algorithmus von Yates fuer den Vektor v
   # ist alles gleich TRUE, so werden alle Schritte
dokumentiert
   # ist alles gleich FALSE, so werden nur die Kontraste
ausgegeben
    n<-length(v)
    k<-log(n,base=2)
    h<-v
    for(i in 1:k)
     { m<-matrix(v,n/2,2,byrow=T)
       v<-c(m[,1]+m[,2],m[,2]-m[,1])
       h<-cbind(h,v)
     }
    if(alles==TRUE) return(h) else return(h[,dim(h)[2]])
 }
```

Die Funktion yates besitzt zwei Argumente. Das erste Argument ist gleich dem Vektor mit den Summen auf den Faktorstufenkombinationen. Mit dem zweiten Argument alles kann man festlegen, ob die Ergebnisse aller Schritte oder nur die Kontraste ausgegeben werden sollen. Standardmäßig werden alle Schritte ausgegeben.

```
> yates(Zeit)

      h  v   v
[1,] 40 86 178
[2,] 46 92  14
[3,] 42  6   6
[4,] 50  8   2

> yates(Zeit,alles=FALSE)

[1] 178  14   6   2
```

Wollen wir mit dem Ergebnis des Algorithmus von Yates die Effekte schätzen und die Quadratsummen bestimmen, so müssen wir es einer Variablen zuweisen und die erste Komponente entfernen.

```
> e<-yates(Zeit,alles=FALSE)
> e<-e[-1]
> e

[1] 14  6  2
```

Die geschätzten Effekte erhalten wir nun dadurch, dass wir den Vektor e durch 2 dividieren

```
> e/2

[1] 7 3 1
```

Für die Quadratsummen müssen wir ihn quadrieren und durch 4 dividieren.

```
> e^2/4

[1] 49  9  1
```

Für den Interaktionsplot zwischen den Faktoren Strecke und Zeitpunkt verwenden wir die Funktion interaction.plot. Diese benötigt drei Argumente. Das erste Argument ist der Faktor auf der Abszisse, das zweite Argument der zweite Faktor und das dritte Argument der vektor mit den Daten. Soll also Strecke der Faktor auf der Abszisse sein, so geben wir ein

```
> interaction.plot(Strecke,Zeitpunkt,Zeit)
```

Soll hingegen der Faktor Zeitpunkt auf der Abszisse stehen, so geben wir ein

```
> interaction.plot(Zeitpunkt,Strecke,Zeit)
```

Schauen wir uns noch das nichtadditive Modell an. Da n=1 ist, können wir nicht testen. Dies zeigt auch die ANOVA-Tabelle:

```
> summary(aov(Zeit~Strecke+Zeitpunkt+Strecke*Zeitpunkt))
                 Df Sum Sq Mean Sq
Strecke           1     49      49
Zeitpunkt         1      9       9
Strecke:Zeitpunkt 1      1       1
```

Wenden wir uns einem Faktoriellen Versuchsplan mit drei Faktoren zu.Beginnen wir wieder mit einem Beispiel.

Es soll untersucht werden, von welchen Faktoren das Erinnerungsvermögen abhängt. Hierzu wurde Personen eine Liste mit 20 dreisilbigen Wörtern vorgelegt oder vorgelesen. Danach hatten die Personen drei Minuten Zeit, alle Wörter aufzuschreiben, die sie sich gemerkt hatten. Es werden die folgenden drei Faktoren betrachtet:

A: Geschlecht mit den Faktorstufen männlich (-) und weiblich (+)
B: Alter mit den Faktorstufen jung (-) und alt (+)
C: Darbietung mit den Faktorstufen akustisch (-) und gedruckt (+)

Auf jeder Faktorstufenkombination wurde ein Versuch durchgeführt und die Anzahl der Wörter bestimmt, die aufgeschrieben wurden.

Hier sind die Daten

```
  A B C Anzahl
  - - -    9
  + - -    9
  - + -    5
  + + -    7
  - - +   10
  + - +   14
  - + +   12
  + + +   10
```

Wir geben die Daten in R ein.

```
> Anzahl<-c(9,9,5,7,10,14,12,10)
```

Damit wir bei den Faktoren nicht so lang Namen haben, nennen wir sie A,B und C. Um sie zu erzeugen, haben wir mehrere Möglichkeiten.

Wir können sie eingeben:

```
> A<-factor(c(-1,1,-1,1,-1,1,-1,1))

> A
[1] -1 1  -1 1  -1 1  -1 1
Levels: -1 1

> B<-factor(c(-1,-1,1,1,-1,-1,1,1))

> B
[1] -1 -1 1  1  -1 -1 1  1
Levels: -1 1

> C<-factor(c(-1,-1,-1,-1,1,1,1,1))

> C
[1] -1 -1 -1 -1 1  1  1  1
Levels: -1 1
```

Wir können die Funktion gen.factorial im Paket AlgDesign verwenden. Dieses Paket muss man zunächst installieren. Hierzu klickt man auf den Schalter

**Packages**

und danach auf

**Install package(s) from CRAN**

Es öffnet sich ein Fenster mit einer Liste, in der man auf AlgDesign klickt. Dann wird das Paket installiert. Dazu muss natürlich eine Verbindung zum Internet vorhanden sein.

Nachdem man

```
> library(AlgDesign)
```

eingegeben hat, kann man die Funktion `gen.factorial` verwenden.

Diese hat vier Argumente. Das erste Argument ist die Anzahl der Faktorstufen bei den Faktoren und das zweite Argument die Anzahl der Faktoren. Das Argument `factors` setzt man auf den Wert `"all"` und mit dem Argument `varNames` kann man den Faktoren Namen geben.

```
> m<-
gen.factorial(2,3,factors="all",varNames=c("Geschlecht","Alter
","Darbietung"))

> m
  Geschlecht Alter Darbietung
1          1     1          1
2          2     1          1
3          1     2          1
4          2     2          1
5          1     1          2
6          2     1          2
7          1     2          2
8          2     2          2
```

Wir sehen, dass die Faktorstufen mit 1 und 2 bezeichnet werden. Wir können auf die Faktoren unter ihren Namen zugreifen, wenn wir

```
> attach(m)
```

eingegeben haben.

```
> Geschlecht
[1] -1 1  -1 1  -1 1  -1 1
Levels: -1 1

> Alter
[1] -1 -1 1  1  -1 -1 1  1
Levels: -1 1

> Darbietung
[1] -1 -1 -1 -1 1  1  1  1
Levels: -1 1
```

Im Folgenden verwenden wir die Namen A, B und C.

Wir können wie bei einem Versuchsplan mit zwei Faktoren die Funktion aov verwenden. Beginnen wir mit dem additiven Modell

```
> summary(aov(Anzahl~A+B+C))
```

```
            Df Sum Sq Mean Sq F value  Pr(>F)
A            1      2       2  0.6667 0.46005
B            1      8       8  2.6667 0.17781
C            1     32      32 10.6667 0.03091 *
Residuals    4     12       3
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1
```

Wir sehen, dass nur C signifikant ist.

Unterstellen wir, dass nur ABC gleich 0 ist, so geben wir ein

```
> summary(aov(Anzahl~A+B+C+A*B+A*C+B*C))

            Df    Sum Sq   Mean Sq   F value Pr(>F)
A            1         2         2      0.25 0.7048
B            1         8         8      1.00 0.5000
C            1        32        32      4.00 0.2952
A:B          1         2         2      0.25 0.7048
A:C          1 3.058e-32 3.058e-32 3.823e-33 1.0000
B:C          1         2         2      0.25 0.7048
Residuals    1         8         8
```

Im vollständigen Modell können wir nicht testen. Wir erhalten die Quadratsummen durch

```
> summary(aov(Anzahl~A+B+C+A*B+A*C+B*C+A*B*C))
            Df    Sum Sq   Mean Sq
A            1         2         2
B            1         8         8
C            1        32        32
A:B          1         2         2
A:C          1 3.058e-32 3.058e-32
B:C          1         2         2
A:B:C        1         8         8
```

Dies erhalten wir aber auch durch

```
> summary(aov(Anzahl~A*B*C))
            Df    Sum Sq   Mean Sq
A            1         2         2
B            1         8         8
C            1        32        32
A:B          1         2         2
A:C          1 3.058e-32 3.058e-32
B:C          1         2         2
A:B:C        1         8         8
```

Das Ergebnis des Algorithmus von Yates liefert der Aufruf

```
> yates(Anzahl)

      h  v  v  v
[1,]  9 18 30 76
[2,]  9 12 46  4
[3,]  5 24  2 -8
[4,]  7 22  2 -4
[5,] 10  0 -6 16
[6,] 14  2 -2  0
[7,] 12  4  2  4
[8,] 10 -2 -6 -8
```

Um die geschätzten Effekte und die Quadratsummen zu erhalten geben wir ein

```
> e<-yates(Anzahl,alles=FALSE)

> e<-e[-1]

> e
[1]  4 -8 -4 16  0  4 -8

> e/4
[1]  1 -2 -1  4  0  1 -2

> e^2/8
[1]  2  8  2 32  0  2  8
```

Wir können natürlich auch die Funktion interaction.plot verwenden, um den Interaktionsplot zwischen zwei Faktoren zu erstellen. Bei dieser werden die Mittelwerte auf den Faktorstufenkombinationen automatisch bestimmt.

Den Interaktionsplot zwischen A und B mit A auf der Abszisse liefert

```
> interaction.plot(A,B,Anzahl)
```

während man den Interaktionsplot zwischen B und C mit C auf der Abszisse folgendermaßen erhält

```
> interaction.plot(C,B,Anzahl)
```

Nun schauen wir uns das Verfahren von Lenth an. Die folgende Funktion ist nur für einen faktoriellen Versuchsplan mit drei Faktoren gedacht. Sie führt das Verfahren auf Basis des ME durch. Alle Faktoren, deren geschätzte Effekte betragsmäßig größer als ME sind, werden als signifikant angesehen.

```
lenth<-function(e)
{ # das Verfahren von LENTH bei 3 Faktoren mit dem ME
  # e ist der Vektor mit den Effekten
    eff<-c("A","B","AB","C","AC","BC","ABC")
    ea<-abs(e)
    o<-order(ea)
```

```
    cat("Die sortierten Effekte\n")
    for (i in 1:7) cat(paste(eff[o][i],ea[o][i],"\n"))
    m1<-median(ea)
    cat(paste("\nM1 =",m1,"\n"))
    s0<-1.5*m1
    cat(paste("s0 =",s0,"\n"))
    m2<-median(ea[ea<2.5*s0])
    cat(paste("M2 =",m2,"\n"))
    PSE<-1.5*m2
    cat(paste("PSE =",PSE,"\n"))
    me<-2.295*PSE
    cat(paste("ME =",me,"\n\n"))
    cat("Signifikante Faktoren:\n")
    cat(eff[ea>me],"\n")
}
```

Wir rufen die Funktion mit den geschätzten Effekten auf:

```
> lenth(yates(Anzahl,alle=FALSE)[-1]/4)

Die sortierten Effekte
AC 0
A 1
AB 1
BC 1
B 2
ABC 2
C 4

M1 = 1
s0 = 1.5
M2 = 1
PSE = 1.5
ME = 3.4425

Signifikante Faktoren:
C
```

Bisher war n=1. Schauen wir uns ein Beispiel mit n=2 an.

Die Studenten Michael Dorin und Stefan Neuhaus ließen in ihrem BI-Projekt Studenten den PRESS-Test durchführen. Bei diesem müssen die Teilnehmer in drei Minuten möglichst viele Aufgaben vom Typ 3-5+1 lösen. Die Zielvariable ist die Anzahl der richtig gelösten Aufgaben. Es wurden zwei Faktoren A und B betrachtet.

A: Schriftgröße mit den Faktorstufen 8 pt (-) und 16 pt (+)
B: Musik mit den Faktorstufen ohne (-) und mit (+)

Hier sind die Daten

```
 A B   Zeit
 - -  42 39
 + -  51 53
 - +  48 43
 + +  87 73
```

Wir weisen die Daten Spalte für Spalte einem Vektor zu:

```
> Zeit2<-c(42,51,48,87,39,53,43,73)
```

Die Faktoren $A$ und $B$ müssen nun die Vorzeichen erhalten, die zu den entsprechenden Komponenten von Zeit2 gehören.

```
> A<-factor(c(-1,1,-1,1,-1,1,-1,1))
> A
[1] -1 1  -1 1  -1 1  -1 1
Levels: -1 1

> B<-factor(c(-1,-1,1,1,-1,-1,1,1))
> B
[1] -1 -1 1  1  -1 -1 1  1
Levels: -1 1
```

Nun gehen wir wie bei n=1 vor. Wir schauen uns das nichtadditive Modell an.

```
> e<-aov(Zeit2~A+B+A:B)
```

Die ANOVA-Tabelle erhalten wir durch

```
> summary(e)
            Df  Sum Sq Mean Sq F value   Pr(>F)
A            1 1058.00 1058.00 36.1709 0.003849 **
B            1  544.50  544.50 18.6154 0.012501 *
A:B          1  264.50  264.50  9.0427 0.039663 *
Residuals    4  117.00   29.25
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1
```

Den Interaktionsplot erhalten wir durch

```
> interaction.plot(B,A,Zeit2)
```

Um die Annahme der Normalverteilung zu überprüfen, erstellen wir eine Normal-Probabilty-Plot der Residuen durch

```
> qqnorm(residuals(e))
> qqline(residuals(e))
```

Um den Algorithmus von Yates durchführen zu können, müssen wir die Summen auf den Faktorstufenkombinationen bestimmen. Hierzu erstellen wir mit der Funktion `matrix` die Matrix der Daten.

```
> m<-matrix(Zeit2,4,2)
> m
     [,1] [,2]
[1,]   42   39
[2,]   51   53
[3,]   48   43
[4,]   87   73
```

```
apply(X, MARGIN, FUN)
```

Dabei ist `X` die Matrix und `MARGIN` die Dimension, auf die Funktion `FUN` angewendet werden soll. Die erste Dimension sind die Zeilen und die zweite die Spalten. Wir geben also ein

```
> apply(m,1,sum)
[1]  81 104  91 160
```

Dieses Ergebnis verwenden wir als Argument der Funktion `yates`. Wir können also eingeben

```
> ey<-yates(apply(matrix(Zeit2,4,2),1,sum))
> ey
       h   v   v
[1,]  81 185 436
[2,] 104 251  92
[3,]  91  23  66
[4,] 160  69  46
```

Wir können nun wieder die geschätzten Effekte und die Quadratsummen bestimmen.

```
> ey<-ey[,3][-1]

> ey
[1] 92 66 46

> ey/4
[1] 23.0 16.5 11.5

> ey^2/8
[1] 1058.0  544.5  264.5
```

Wenden wir uns einem faktoriellen Versuchsplan mit drei Faktoren und n=2 zu.

An der Kasse bei ALDI Es wurde die Wartezeit (in Sekunden) bestimmt. Dabei wurden die folgenden Faktoren betrachtet.

A: Einkaufstag mit den Faktorstufen Mittwoch (-) und Samstag (+)
B: Uhrzeit mit den Faktorstufen Vormittag (-) und Mittag (+)
C: Filiale mit den Faktorstufen stadtnah (-) und außerhalb (+)

Hier sind die Daten

```
  A B C    Zeit
  - - -  366 257
  + - -  312 255
  - + -  405 453
  + + -  321 317
  - - +  456 508
  + - +  322 353
  - + +  382 461
  + + +  332 363
```

Wir geben die Daten in R ein.

```
> aldi<-
c(366,312,405,321,456,322,382,332,257,225,453,317,508,353,461,
363)
```

Um die Faktoren zu erzeugen, haben wir mehrere Möglichkeiten.

Wir können sie eingeben. Dabei können wir von der Funktion `rep` Gebrauch machen. Sie hat die Argument `x` und `times`. Dabei ist `x` ein Vektor der Länge k. Ist `times` eine Zahl, so wird der Vektor `x` times-mal wiederholt.

```
> rep(c(366,312,405),2)
[1] 366 312 405 366 312 405
```

Ist `times` hingegen ein Vektor der Länge k, so wird die i-te Komponente von x times[i]-mal wiederholt.

```
> rep(c(366,312,405),c(1,2,3))
[1] 366 312 312 405 405 405
```

Wir geben also ein:

```
> A<-factor(rep(rep(c(-1,1),4),2))
> A
 [1] -1 1  -1 1  -1 1  -1 1  -1 1  -1 1  -1 1  -1 1
Levels: -1 1
```

```
> B<-factor(rep(rep(c(-1,1),c(2,2)),4))
> B
 [1] -1 -1 1  1  -1 -1 1  1  -1 -1 1  1  -1 -1 1  1
Levels: -1 1

> C<-factor(rep(rep(c(-1,1),c(4,4)),2))
> C
 [1] -1 -1 -1 -1 1  1  1  1  -1 -1 -1 -1 1  1  1  1
Levels: -1 1
```

Wir können aber auch die Funktion `gen.factorial` im Paket `AlgDesign` verwenden, um die Faktoren zu erzeugen. Diese hat vier Argumente. Das erste Argument `levels` ist die Anzahl der Faktorstufen der Faktoren und das zweite Argument `nVars` die Anzahl der Faktoren. Das Argument `factors` setzt man auf den Wert `"all"` und mit dem Argument `varNames` kann man den Faktoren Namen geben.

Wir geben ein

```
> m<-
gen.factorial(2,3,factors="all",varNames=c("Tag","Uhrzeit","Fi
liale"))

> m
  Tag Uhrzeit Filiale
1   1       1       1
2   2       1       1
3   1       2       1
4   2       2       1
5   1       1       2
6   2       1       2
7   1       2       2
8   2       2       2
```

Wir sehen, dass die Faktorstufen mit 1 und 2 bezeichnet werden.Wir können auf die Faktoren unter ihren Namen zugreifen, wenn wir

```
> attach(m)
```

eingegeben haben.

```
> Tag
[1] 1 2 1 2 1 2 1 2
Levels: 1 2


> Uhrzeit
```

```
[1] 1 1 2 2 1 1 2 2
Levels: 1 2


> Filiale
[1] 1 1 1 1 2 2 2 2
Levels: 1 2
```

Da n=2 ist, müssen wir nun die Matrix m verdoppeln. Hierzu benutzen wir die Funktion rbind.

```
> m<-rbind(m,m)
> m
   Tag Uhrzeit Filiale
1    1       1       1
2    2       1       1
3    1       2       1
4    2       2       1
5    1       1       2
6    2       1       2
7    1       2       2
8    2       2       2
11   1       1       1
21   2       1       1
31   1       2       1
41   2       2       1
51   1       1       2
61   2       1       2
71   1       2       2
81   2       2       2


> attach(m)
```

Zur Erstellung der ANOVA-Tabelle verwenden wir wie bei einem Versuchsplan mit zwei Faktoren die Funktion aov.
Beginnen wir mit dem additiven Modell

```
> e<-aov(aldi~Tag+Uhrzeit+Filiale)


> summary(e)
            Df Sum Sq Mean Sq F value  Pr(>F)
Tag          1  34503   34503 12.1574 0.00449 **
Uhrzeit      1   3452    3452  1.2162 0.29174
Filiale      1  16965   16965  5.9778 0.03088 *
Residuals   12  34056    2838
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1
```

Wir sehen, dass die Faktoren Tag und Filiale signifikant sind.

Um die Annahme der Normalverteilung zu überprüfen, erstellen wir eine Normal-Probabilty-Plot der Residuen durch

```
> qqnorm(residuals(e))
> qqline(residuals(e))
```

Wir können aber auch Tests auf Normalverteilung durchführen.

Der Aufruf

```
>
ks.test(residuals(e),"pnorm",mean(residuals(e)),sd(residuals(e
)))
```

liefert das Ergebnis des Kolmogorow-Tests auf Normalverteilung.

```
     One-sample Kolmogorov-Smirnov test

data:  residuals(e)
D = 0.1676, p-value = 0.7596
alternative hypothesis: two.sided
```

Besser ist aber der Shapiro-Wilk-Test

```
> shapiro.test(residuals(e))

        Shapiro-Wilk normality test

data:  residuals(e)
W = 0.9562, p-value = 0.593
```

Beide Tests lehnen die Annahme derNormalverteilung zum Signifikanzniveau 0.05 nicht ab.

Schauen wir uns noch das nichtadditive Modell an. Bei diesem müssen wir nicht alle Faktorstufenkombinationen eingeben. Der
höchste Interaktionseffekt reicht aus.

```
> e<-aov(aldi~Tag*Uhrzeit*Filiale)
```

```
> summary(e)
                   Df Sum Sq Mean Sq F value   Pr(>F)
Tag                 1  34503   34503 16.9148 0.003378 **
Uhrzeit             1   3452    3452  1.6921 0.229540
Filiale             1  16965   16965  8.3170 0.020389 *
Tag:Uhrzeit         1      3       3  0.0015 0.970041
Tag:Filiale         1   1073    1073  0.5258 0.489036
Uhrzeit:Filiale     1  11936   11936  5.8513 0.041919 *
Tag:Uhrzeit:Filiale 1   4727    4727  2.3172 0.166451
Residuals           8  16319    2040
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1
```

Wir können natürlich auch den Algorithmus von Yates mit der Funktion `yates` durchführen. Hierzu müssen wir den Vektor `aldi` in eine Matrix transfomieren und die Zeilensummen bilden.

```
> e<-yates(apply(matrix(aldi,8,2),1,sum))
> e
       h    v    v    v
[1,] 623 1160 2656 5833
[2,] 537 1496 3177 -743
[3,] 858 1639 -306  235
[4,] 638 1538 -437    7
[5,] 964  -86  336  521
[6,] 675 -220 -101 -131
[7,] 843 -289 -134 -437
[8,] 695 -148  141  275
```