1 Einleitung

1.1 Was ist R?

R ist ein Programm-System für Statistik (und Verwandtes).

S ist die Sprache, die auch S-Plus zugrunde liegt.

S-Plus ist kommerziell und arbeitet vor allem mit Menus.

R ist free software und basiert auf Kommandos.

Bisher gibt es - ausser dem R-Commander - kein (offizielles) Menü-System.

Menüs haben Nachteile: Nachvollziehbarkeit.

Gespeicherte Kommandos dokumentieren Analyse und geben einem die Möglichkeit, die Analyse nochmals mit leicht veränderten Daten / Optionen durchzuführen.

1.2 R Software

R

- Homepage: `http://www.r-project.org`
- Software: `http://stat.ethz.ch/CRAN`

Unterstützte Betriebssysteme:

Linux, Mac, Windows

Texteditoren

- WinEdt: `http://www.winedt.com/`
- Emacs: `http://www.gnu.org/software/emacs/`
  ESS: `http://stat.ethz.ch/ESS/`
- Tinn-R: `http://www.sciviews.org/Tinn-R/`

2 Grundlagen

2.1 Einführende Beispiele

a Daten, data.frame

Datenmatrix: `data.frame`.

Variablen: numerisch (quantitativ) oder nominal (kategorial)

b Daten einlesen.

```
d.sport  <- read.table(
"http://stat.ethz.ch/Teaching/Datasets/
NDK/sport.dat", header=TRUE)
```

```
> d.sport
            weit kugel hoch  disc stab speer punkte
OBRIEN      7.57 15.66  207 48.78  500 66.90   8824
BUSEMANN    8.07 13.60  204 45.04  480 66.86   8706
DVORAK      7.60 15.82  198 46.28  470 70.16   8664
:           :    :      :   :      :   :       :
:           :    :      :   :      :   :       :
:           :    :      :   :      :   :       :
CHMARA      7.75 14.51  210 42.60  490 54.84   8249
```

Eine Variable auswählen:

`> d.sport[,2]` oder `d.sport[,"kugel"]` oder

`> d.sport$kugel`

Eine Beobachtung auswählen:

`> d.sport[1,]` oder `d.sport["OBRIEN",]`

Histogramm:

`> hist(d.sport[,"kugel"])`

Grafik-Fenster wird in R automatisch geöffnet.

c **Funktions-Aufruf.**

Funktionen haben obligatorische Argumente und freiwillige.

`> hist(d.sport[,"kugel"], nclass=10)`

`nclass` freiwilliges Arg. Histogramm mit ungefähr 10 Klassen.

Keine obligatorischen Argumente $\longrightarrow$ leere Klammern!

`> objects()` (oder `ls()`)

`q()` : Session beenden.

Ohne Klammern: Programmcode der Funktion wird ausgegeben!

Die Argumente einer Funktion haben eine bestimmte Reihenfolge
und jedes hat auch einen Namen.

`> hist(x=d.sport[,"kugel"], nclass=10)`

Für `nclass` Namen angeben oder auswendig lernen,
dass `nclass` das 17. Argument ist.

`> hist(d.sport[,"kugel"],,,,,,,,,,,,,,,,10)`

Hilfe:

`> help(hist)` oder `?hist`

2.1

d **Streudiagramme.**

$$> \texttt{plot(}\underbrace{\texttt{d.sport[,"kugel"]}}_{x-Koordinaten}\texttt{,}\underbrace{\texttt{d.sport[,"speer"]}}_{y-Koordinaten}\texttt{)}$$

Viele freiwillige Argumente!

e **Streudiagramm-Matrix.**

`> pairs(d.sport)`

jede Variable der Datenmatrix `d.sport` wird gegen jede aufgezeichnet.

2.1

f Grafik drucken.

```
> dev.print()
```

Windows: Drucker-Symbol klicken (ev. zuerst mit der Maus ins Graphikfenster klicken, um es zu aktivieren).

g Grafik als .ps-file abspeichern.

```
> dev.print(device=postscript,
      file="C://graphic.ps")
```


h Session beenden.

```
> q()
```

Workspace sichern?
Wenn Sie "yes" antworten, werden Ihnen in der nächsten Session alle „Objekte“, die Sie in dieser Session erzeugt haben, wieder zur Verfügung stehen.

2.2 Namensgebung

a Namen.

Buchstabe, dann [Buchstabe, Ziffer, "."] beliebig oft.
case sensitive!
Keine "reservierten" Namen benutzen. (z.B. mean, data.frame)

b Vorschlag:

Die Namen aller Objekte, die Sie erzeugen, beginnen mit 1 Buchstaben, gefolgt von einem Punkt, gefolgt von weiteren Buchstaben und Ziffern
Beginn: Art des Objekts.
`d.sport`: Daten
`t.v`: temporär
`r.regr`: Resultate
`f.maxi`: Eigene Funktionen

2.3 Vektoren

```
> t.v  <- d.sport[,"kugel"]
> t.v

[1] 15.66 13.60 15.82 15.31 16.32 ...
[13] 16.97 14.69 14.51

> t.a  <- c(3.1, 5, -0.7, 0.9, 1.7)
```

a Alphanumerische Vektoren.

```
> t.b  <- c("Andi", "Bettina", "Christian")
```

```
> seq(0,3,by=0.5)
[1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0

> 1:9
[1] 1 2 3 4 5 6 7 8 9

> rep(0.7,5)
[1] 0.7 0.7 0.7 0.7 0.7

> rep(c(1,3,5),length=8)
[1] 1 3 5 1 3 5 1 3
```


Wichtige, einfache Funktionen für Vektoren.

Aufruf, Beispiel	Bedeutung
`length(t.v)`	Länge, Anzahl Elemente
`sum(t.v)`	Summe aller Elemente
`mean(t.v)`	arithmetisches Mittel der Elemente
`var(t.v)`	empirische Varianz
`range(t.v)`	Wertebereich

b Viele gleiche Rechnungen.

```
> apply(d.sport,2,mean)
 weit   kugel    hoch    disc    stab   speer   punkte
7.597  15.199 202.000  46.376 498.000 61.995 8444.667
```

Wenn Sie eine „konventionelle“ Programmiersprache kennen, würden Sie solche Aufgaben wohl über eine Schleife (loop) lösen.
Später werden wir solche Programmierelemente einführen – und gleichzeitig „predigen“, dass man sie vermeiden soll!

2.3

c Logische Vektoren.

```
> (1:5)>=3
[1]  FALSE  FALSE  TRUE  TRUE  TRUE
```

Vergleichsoperationen `<, <=, >, >=, ==, !=`

Operationen `&` (und), `|` (oder), `!` (nicht).

```
> t.i  <- (t.a>2)&(t.a<5)
> t.a[t.i]
[1] 3.1
```

2.4 Arithmetik

```
> 2+5
[1] 7
```

+ , − , ∗ , / ,ˆ.

Auf Vektoren werden die Operationen **elementweise** angewandt.

```
> (2:5) ^ c(2,3,1,0)
[1]  4 27 4 1
```

Prioritäten der Operationen wie üblich. Klammern setzen!

```
> (2:5) ^ 2
[1]  4  9 16  25
```


Elemente werden zyklisch wieder verwendet:

```
> (1:5)-(0:1)
[1] 1 1 3 3 5
```

```
Warning message:
longer object length is not a multiple of
shorter object length in: (1:5) - (0:1)
```

```
> (1:6)-(0:1)
[1] 1 1 3 3 5 5
```

Achtung: Hier gibt R keine Warnung aus!

2.5 Elemente auswählen

- Indices (ganze Zahlen):

```
> t.v[c(1,3,5)]
[1] 15.66 15.82 16.32
> d.sport[c(1,3,5),1:3]
```

```
           weit kugel hoch
OBRIEN     7.57 15.66  207
DVORAK     7.60 15.82  198
HAMALAINEN 7.48 16.32  198
```


- Bei data.frames: Namen der Zeilen oder Spalten

```
> d.sport[c("OBRIEN","DVORAK"),
c("kugel","speer","punkte")]
```

```
       kugel speer punkte
OBRIEN 15.66 66.90   8824
DVORAK 15.82 70.16   8664
```

- Logische Vektoren

```
> t.a[c(TRUE,FALSE,TRUE,TRUE,FALSE,FALSE)]

[1] 3.1 -0.7 0.9

> d.sport[d.sport$kugel > 16, c(2,7)]

           kugel punkte
HAMALAINEN 16.32   8613
PENALVER   16.91   8307
SMITH      16.97   8271
```


2.6 Verteilungen und Zufallszahlen

- Wahrscheinlichkeiten:

```
> dnorm(0.5, mean=0, sd=1)
[1] 0.35207
```

- Kumulative Verteilungsfunktionen

```
> pnorm(1.96, mean=0, sd=1)
[1] 0.975
```

- Quantile

```
> qnorm(0.9, mean=0, sd=1)
[1] 1.28155
```


- Zufallszahlen:

```
> rnorm(5, mean=2, sd=2)
[1] 2.964 2.730 -2.581 -2.260 -0.781
```

Poissonverteilung: `dpois, ppois, qpois, rpois`

```
> rpois(20, lambda=3.5)
```

Namensgebung:

- Die Abkürzungen der Verteilungsnamen sind ziemlich willkürlich. Die Autoren waren wohl etwas schreibfaul.

Diskrete Verteilungen	
`binom`	Binomial-Verteilung
`pois`	Poisson-Verteilung
`hyper`	hypergeometrische Verteilung
Stetige Verteilungen	
`unif`	uniforme Verteilung
`norm`	Normalverteilung
`lnorm`	Log-Normalverteilung
`t, F, chisq`	t-, F-, χ^2- (Chiquadrat-)Verteilung
`weibull, gamma`	Weibull-, Gamma-Verteilung

2.6

b Zufallszahlen.

```
> runif(4, min=0, max=1)
[1] 0.81341 0.04849 0.17556 0.02286
> runif(2, min=0, max=1)
[1] 0.1241 0.2918

> set.seed(27)
> runif(1, min=0, max=1)
[1] 0.9717502
> set.seed(27)
> runif(1, min=0, max=1)
[1] 0.9717502
```


c Darstellung von Verteilungen.

Diskrete Verteilungen mit:

```
> plot(0:15,dpois(0:15,lambda=3.5),
   type="h")
```

Stetige Verteilungen mit:

```
> curve(dnorm(x,5,2),xlim=c(-1,10),xlab="x",
   ylab="Dichte",main="Normalverteilung")
```


2.7 Einfache Statistik-Funktionen

a Tests und Vertrauensintervalle.

```
> binom.test(3, 20, p=0.4)

               Exact binomial test
data:  3 and 20
number of successes = 3, number of trials = 20,
p-value = 0.02243
alternative hypothesis: true probability
   of success is not equal to 0.4
95 percent confidence interval:
   0.0321 0.3789
```



```
> t.hh  <- d.sport[,"hoch"]>200
> t.kugel  <- d.sport[,"kugel"]
> wilcox.test(t.kugel[t.hh],t.kugel[!t.hh])


          Wilcoxon rank sum test

   data:  t.kugel[t.hh] and t.kugel[!t.hh]
   W = 20, p-value = 0.4559
   alternative hypothesis: true mu is
      not equal to 0
```

```
> t.test(t.kugel[t.hh],t.kugel[!t.hh],
   var.equal=TRUE)
          Two Sample t-test
  data:  t.kugel[t.hh] and t.kugel[!t.hh]
  t = -0.8066, df = 13, p-value = 0.4344
  alternative hypothesis: true difference
     in means is not equal to 0
  95 percent confidence interval:
   -1.694  0.773
  sample estimates:
  mean of x mean of y
      15.01     15.47
```


2.7

b Grafische Darstellungen für zwei Stichproben.

```
> boxplot(t.kugel ~ t.hh, names=c("<= 200",
    "> 200"), xlab="Hochsprung", ylab="Kugel"))

> t.h1 <- hist(t.kugel[t.hh],breaks=13:17,
   plot=FALSE)
> t.h2 <- hist(t.kugel[!t.hh], breaks=13:17,
   plot=FALSE)
> barplot(rbind(t.h1$density,t.h2$density),
   beside=TRUE, names.arg=t.h1$mid)
```


2.8 Fehlende Daten

```
> t.kugel[t.kugel < 14]  <- NA
> t.kugel
[1] 15.66 NA 15.82 15.31 16.32 14.01 NA ...

> t.kugel == NA
[1] NA NA NA NA NA ...
```

ergibt nichts Vernünftiges, stattdessen:

```
> is.na(t.kugel)
[1] FALSE TRUE FALSE FALSE FALSE FALSE TRUE
...
```



```
> mean(t.kugel)
[1] NA

> mean(t.kugel, na.rm=TRUE)
[1] 15.45
```

Bei „höheren" Funktionen (z.B. `wilcox.test` und `t.test`) kann mit dem Argument `na.action` gewählt werden, wie fehlende Werte behandelt werden.

`na.action=na.omit`: fehlende Werte weglassen

`na.action=na.pass`: gibt Objekt ungeändert zurück

etc.

3 Grundelemente der Sprache S

3.1 Funktionen selber schreiben

a Beispiel Maximum eines Vektors bestimmen und Index ausgeben.

```
f.maxi <- function(data){
  t.max <- max(data, na.rm=TRUE)
  t.i <- match(t.max, data)
  return(c(max=t.max, i=t.i))
}
```

```
  > f.maxi(c(3,4,78,2))
    > max   i
       78   3
```


Diese Funktionen können nun z.B. in

```
apply
```

benutzt werden.

```
apply(d.sport, 2, f.maxi)
```

```
>   weit kugel hoch  disc stab speer punkte
max 8.07 16.97  213 49.84  540 70.16   8824
i   2.00 13.00    8  4.00    6  3.00      1
```


3.2 Steuerbefehle

a `if, else`

Berechne die Logarithmen eines Vektors.

```
f.log <- function(x){
  if (any(x <= 0)) {
    stop ("Mindestens eine Zahl ist kleiner gleich 0")
  } else {
    t.x <- log(x)
    return(t.x)
  }
}
```

"`}`" und `else` müssen auf die selbe Zeile!

b `for`

Berechne die ersten zwölf Fibonacci-Zahlen.

```
fib <- c(1,1)

for (i in 1:10){
  fib <- c(fib, fib[i]+fib[i+1])
}

> fib
  [1]   1   1   2   3   5   8   13   21   34   55   89   144
```

$\longrightarrow$ for-Schlaufen sollten wenn möglich vermieden werden!

c Weitere Steuerbefehle: `while, repeat (& break)`

3.3 Fehlermeldungen

Fehlermeldungen sind manchmal schwer verständlich. Die S-Sprache ist so flexibel, dass sie trotz Fehler im Code weiterrechnet (evt. mit NA's) und dann erst später auf einen unlösbaren Befehl stösst. Dieser Fehler wird dann ausgegeben und lässt Sie am falschen Ort nach der Fehlerquelle suchen.

Da Funktionen weitere Funktionen aufrufen steht man vor der Frage, wo der Fehler nun wirklich passiert ist. Antworten findet man mit:

```
traceback()
```

oder

```
debug(function)
```

"rausspringen" mit Q

```
undebug(function)
```


3.4 Objekte

a Listen. Listen sind Zusammenfassungen verschiedenster Objekte zu einem "Superobjekt".

```
t.list <- list(V1 = c(1,2,3), V2 = "Hallo Welt",
               V3 = 4, V4 = list(1,"Stat"))
```

Auf ein Element der Liste `t.list` greift man zu mit

```
t.list[[1]]
```

oder - falls das Element einen Namen hat, z.B. V1 -

```
t.list$V1
```


b Dataframes Übliche Art, Daten anzugeben. Datentypen wild gemischt.

```
t.alter <- c(24, 15, 19, 22)
t.sex <- rep(c("Mann", "Frau"), 2)
t.krank <- c(TRUE, FALSE, FALSE, FALSE)

d.bsp <- data.frame(x=t.alter, y=t.sex, z=t.krank)
rownames(d.bsp) <- c("Urs","Anna","Max","Trudi")


d.weithoch <- d.sport[,c("weit", "hoch")]
```

c **Matrizen**

Vereinfachte Version von Dataframes. Nur Daten vom gleichen Typ.

```
t.m1 <- matrix(1:15, nrow=3, ncol=5)

     [,1] [,2] [,3] [,4] [,5]
[1,]    1    4    7   10   13
[2,]    2    5    8   11   14
[3,]    3    6    9   12   15

t.m2 <- matrix(1:15, nrow=3, byrow=TRUE)
```

Auswahl einzelner Elemente wie üblich:

```
> t.m1[2,1:3]
[1] 2 5 8
```


d **str()**

Gibt einem "alles Mögliche" an Informationen über ein Objekt aus.

```
> str(d.sport)
'data.frame':   15 obs. of  7 variables:
 $ weit  : num  7.57 8.07 7.6 7.77 7.48 ...
 $ kugel : num  15.7 13.6 15.8 15.3 16.3 ...
 $ hoch  : int  207 204 198 204 198 201 ...
 $ disc  : num  48.8 45.0 46.3 49.8 49.6 ...
 $ stab  : int  500 480 470 510 500 540 ...
 $ speer : num  66.9 66.9 70.2 65.7 57.7 ...
 $ punkte: int  8824 8706 8664 8644 8613 ...
```


e **Abspeichern und Laden**

Objekte können abgespeichert werden mit:

```
> save(d.sport, t.v, file = "C://sport.RData")
```

Und wieder geladen werden mit:

```
> load(file = "C://sport.RData")
```


3.5 Packages

Funktionen sind in Packages gespeichert, von denen die meisten beim Aufstarten von R nicht geladen werden.
Vorteile:

- schnell
- schlank

Die Packages mit den Basisbefehlen werden zu Beginn geladen (z.B. `base`).

a Wie finde ich eine benötigte Funktion und das Package dazu?

- in R: `help.search("...")`
- `http://www.r-project.org/`
- `http://www.google.com/`

b Laden und Inhalt eines Package

```
library(MASS)

library(help=MASS)
```


4 Lineare Regression

4.1 Korrelation

a Neue Daten einlesen:

```
> d.basisch  <- read.table(
  "http://stat.ethz.ch/Teaching/Datasets/
  basischOhneNA.dat", header=TRUE)
```



```
> d.basisch
```

```
       ph   l.sar  height h.quad
1   7.325  0.0969   5.91 34.9281
2   7.690  0.4393   5.20 27.0400
3   7.900  1.0000   4.40 19.3600
:       :       :      :      :
:       :       :      :      :
:       :       :      :      :
123 8.475  1.3334   1.82  3.3124
```

Zielvariable: Baumhöhen (height) oder quadrierte Baumhöhen (h.quad).

Erklärende Variablen: PH-Wert von Böden,
l.sar (= log(sodium absorbtion ratio))

Berechne die Pearson-Korrelation von ph und height

```
cor(d.basisch[,"ph"], d.basisch[,"height"])
```

Bild dazu betrachten

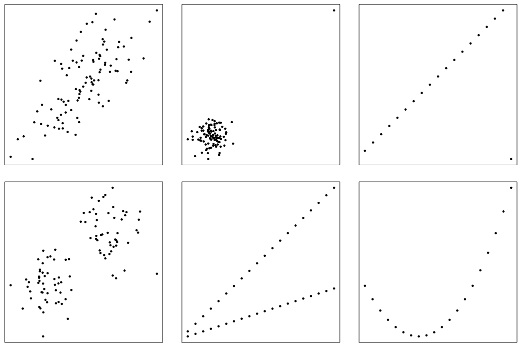
```
plot(d.basisch[,"ph"], d.basisch[,"height"])
```

Andere Korrelationsarten

```
cor(x, y, method = "spearman")
cor(x, y, method = "kendall")
```

Aber ACHTUNG! bei Korrelationen:

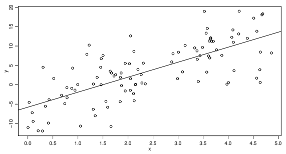
Alle Bilder zeigen 2 Variablen mit einer Korrelation von 0.7!



(entnommen aus Skript "Regression" von Prof. H.R. Künsch)

4.2 Einfache lineare Regression

Regression ist die am meisten verbreitete Methodik der Statistik.



a Lineares Modell $Y_i = \alpha + \beta x_i + E_i, \quad E_i \sim \mathcal{N}(0, \sigma^2)$

an Daten anpassen mit Befehl `lm`

```
fit <- lm(height ~ ph, data = d.basisch)
```

(für die Modellschreibweise siehe Skript Kapitel 5.2)

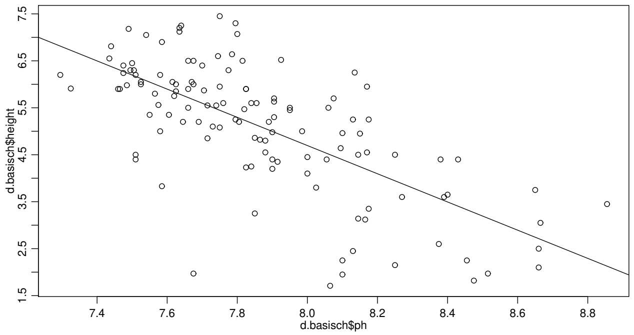
b Zusammenfassung des Objekts `fit` anzeigen.

```
summary(fit)
```

```
Call:
lm(formula = height ~ ph, data = d.basisch)

Residuals:
   Min       1Q   Median       3Q      Max
-3.70195 -0.54712  0.08745  0.66626  2.00330

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  28.7227     2.2395   12.82   <2e-16 ***
ph           -3.0034     0.2844  -10.56   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1

Residual standard error: 1.008 on 121 degrees of freedom
Multiple R-Squared: 0.4797,   Adjusted R-squared: 0.4754
F-statistic: 111.5 on 1 and 121 DF,  p-value: < 2.2e-16
```



```
> plot(d.basisch$ph, d.basisch$height)
> abline(fit)
```



c Residuenanalyse

```
plot(fit)
```

ergibt vier Plots:

- Tukey-Anscombe Plot (Ist die Varianz der Fehler E_i in etwa konstant?)
- Q-Q Plot (Sind die Fehler E_i in etwa normalverteilt?)
- Scale Location Plot (In etwa analog zu Tukey-Anscombe Plot.)
- Leverage Plot (Welche Beobachtungen haben grossen Einfluss auf die angepasste Regressionsgerade?)

Was steckt in diesem `fit` alles drin?

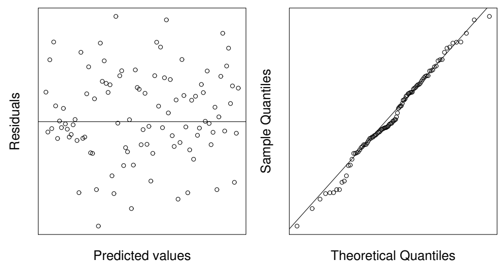
```
str(fit)

List of 12
 $ coefficients : Named num [1:2] 28.72 -3.00
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "ph"
 $ residuals    : Named num [1:123] -0.813 -0.427 -0.596 ...
  ..- attr(*, "names")= chr [1:123] "1" "2" "3" "4" ...

    :     :
    :     :
 $ fitted.values: Named num [1:123] 6.72 5.63 5.00 4.26 5.85 ...
  ..- attr(*, "names")= chr [1:123] "1" "2" "3" "4" ...

    :     :
    :     :
```


Residuenplots "von Hand":

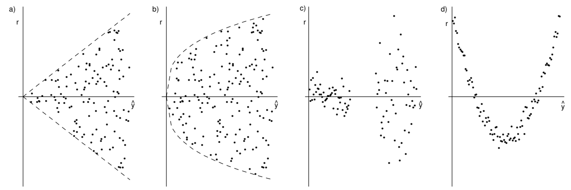
`plot(fit$fitted, fit$resid)` → Tukey-Anscombe Plot

`qqnorm(fit$resid)` → Quantil-Quantil Plot

`qqline(fit$resid)` → passt Gerade an



Verschiedene Tukey-Anscombe Bilder:



(entnommen aus Skript "Regression" von Prof. H.R. Künsch)

4.3 Multiple lineare Regression

a Lineares Modell

$$Y_i = \alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + E_i \qquad E_i \sim \mathcal{N}(0, \sigma^2)$$

an Daten anpassen mit Befehl `lm`:

```
fitm <- lm(height ~ ph + l.sar, data = d.basisch)
```



```
summary(fitm)
```

```
Call:
lm(formula = height ~ ph + l.sar, data = basisch)

Residuals:
     Min       1Q   Median       3Q      Max
-4.13139 -0.49115  0.08488  0.64875  2.47543

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  26.9466     2.7445   9.818  < 2e-16 ***
ph           -2.7558     0.3603  -7.649  5.6e-12 ***
l.sar        -0.2519     0.2255  -1.117    0.266
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1

Residual standard error: 1.007 on 120 degrees of freedom
Multiple R-Squared: 0.485,      Adjusted R-squared: 0.4764
F-statistic: 56.51 on 2 and 120 DF,  p-value: < 2.2e-16
```


b Residuenanalyse

Die selben Plots wie vorher.

```
plot(fitm)
```

Evt. noch Residuen gegen die einzelnen Variablen zeichnen.

```
plot(d.basisch$ph, fitm$resid)
plot(d.basisch$l.sar, fitm$resid)
```