

Einführung in R

Für neue Anwender und Umsteiger anderer
Statistik Software

Jan Marvin Garbuszus und Sebastian Jeworutzki

26. Oktober 2016

UseR Bochum/Ruhr



- Open-Source-Software für statistische Datenverarbeitung
- **R** ist eine Programmiersprache. Auswertungen über Skripte
- Für **R** gibt es verschiedene Entwicklungsumgebungen (RStudio, RKWard)

Warum R?

- R ist kostenlos (download)
- R gibt es für Windows-, Mac- und Unix-Systeme
- Qualitätssicherung durch Kern-Entwicklerteam
- R kann um viele zusätzliche Funktionen erweitert werden (CRAN/R-Pakte)
- R hat Schnittstellen zu vielen anderen Anwendungen (SPSS, $\text{\LaTeX}$,...)
- R fördert reproduzierbare Datenanalyse (Skripte und Möglichkeit zum "Literal Programming")
- R ist objektorientiert, d. h. einfache Verknüpfung mehrerer Analyseschritte ist möglich
- Umfangreiche Möglichkeiten bei der Erstellung von Grafiken

- Für Nutzer die nur grafische Oberflächen (etwa in Excel und SPSS) gewohnt sind, eine zunächst ungewohnte Bedienung
- Einarbeitung ist durch Syntaxorientierung langwieriger
- Dokumentation ist verglichen mit bspw. Stata wesentlich weniger umfangreich
- Einzelne Funktionen, insbesondere solche, die aus zusätzlichen Paketen stammen, sind weniger standardisiert als in anderen Statistikprogrammen

Was wollen wir machen?

- Jemand hat Daten erhoben und wir dürfen damit arbeiten
- Wir wollen diese Daten auswerten
 - Dazu nutzen wir deskriptive und
 - multivariate Verfahren

- Unsere Daten sind in diesem Fall Sekundärdaten
- Diese wurden nicht von uns erhoben
- Wir werden sie auswerten
- Die Daten sind in elektronischer Form eines Datensatzes bereitgestellt
- Ein solcher Datensatz ist z. B. der Allbus

- Wenn wir einen Datensatz haben, dann sind die Daten darin in einer Tabelle angeordnet
- In der obersten Zeile steht der Name einer jeden Spalte, darunter folgen Zeilenweise die einzelnen Beobachtungen

- Ein Beispiel für einen solchen Datensatz:

Studis

Name	Alter	Semesterzahl
Pit	23	5
Claudia	22	3
Marta	22	4

R

- Zunächst RStudio starten
- RStudio ist eine Oberfläche für R
- Sowohl R als auch RStudio gibt es kostenlos für Windows, Mac OS X und Linux zum Download unter <https://cran.r-project.org> und <https://www.rstudio.org>
- Auf den Computern im CIP Raum ist beides installiert

```
R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"  
Copyright (C) 2016 The R Foundation for Statistical Computing  
Platform: x86_64-pc-linux-gnu (64-bit)
```

```
R ist freie Software und kommt OHNE JEGLICHE GARANTIE.  
Sie sind eingeladen, es unter bestimmten Bedingungen weiter zu verbreiten.  
Tippen Sie 'license()' or 'licence()' für Details dazu.
```

```
R ist ein Gemeinschaftsprojekt mit vielen Beitragenden.  
Tippen Sie 'contributors()' für mehr Information und 'citation()',  
um zu erfahren, wie R oder R packages in Publikationen zitiert werden können.
```

```
Tippen Sie 'demo()' für einige Demos, 'help()' für on-line Hilfe, oder  
'help.start()' für eine HTML Browserschnittstelle zur Hilfe.  
Tippen Sie 'q()', um R zu verlassen.
```

```
>
```

- Am Ende der Konsole steht jetzt das größer Zeichen



>

- Ab sofort nennen wir das die Eingabezeile
- Alles was R ausführen soll, wird hier rein gepackt
- Das macht RStudio für uns
- Wir schreiben in unser Syntaxfenster und RStudio kopiert die Befehle in die Kommandozeile

- R ist ein Statistik Programm, also rechnen wir damit erst einmal

```
# Grundrechenarten
```

```
1+1
```

```
1*2
```

```
12-13
```

```
10/2
```

```
# Wie in Stata ist der Punkt das Dezimaltrennzeichen
```

```
10.3 + 5
```

- Soweit so gut

```
# Klammern werden zuerst ausgewertet
(2+3)*2

# Potenz "2 hoch 2"
2^2

# Wurzel aus vier ziehen
sqrt(4)

# oder
4^(1/2)
```

- Wer bereits mit Stata gearbeitet hat wird erkennen, dass die Befehle bisher ganz analog zur Stata Sprache sind

And then Satan said, “put the
alphabet in math”.

- Wir wollen Buchstaben nutzen, um uns das Leben zu erleichtern
- Nehmen wir das Alter der Studis aus dem Beispieldatensatz

Daten

Name	Alter	Semesterzahl
Pit	23	5
Claudia	22	3
Marta	22	4

```
# Das durchschnittliche Alter betraegt  
(23 + 22 + 22)/3
```

- Das ist viel Tipparbeit, schreiben wir also das Alter in ein Objekt

```
# Objekte bestehen aus Objekten, hier Zahlen 23 und 2x22.  
# Mit c() werden sie zu einem Vektor zusammengefügt.  
alter <- c(23, 22, 22)
```

- Jetzt gibt es ein Objekt alter. Damit können wir arbeiten und uns z. B. das mittlere Alter ausgeben lassen

```
mean(alter)
```

- Wir haben gerade `<-` genutzt, um etwas in ein Objekt zu schreiben
- Neben dem Zuweisungspfeil kann man auch das einfache Gleichheitszeichen nutzen

```
# beides funktioniert  
a <- 1  
a = 1
```

Da Gleichheitszeichen aber auch für logische Abfragen genutzt werden, sollte man den Zuweisungspfeil nutzen `a=1` sieht schnell aus wie `a==1`.

- Konstruieren wir uns noch ein Objekt aus den Namen der Studis

```
# Worte werden in Anführungszeichen gesetzt  
namen <- c("Pit", "Claudia", "Marta")  
  
# Jetzt noch die Semesterzahl  
semester <- c(5, 3, 4)
```

- Jetzt haben wir drei Objekte (Vektoren)

In RStudio werden Objekte im Reiter 'Environment' angezeigt.

- Diese Objekte fassen wir jetzt zu einem neuem Objekt, einem Datensatz zusammen
- Beim zusammenfassen können wir Variablennamen vergeben

```
# Die Variablen im Datensatz heissen wie unsere Vektoren
studis <- data.frame(namen=namen,
                     alter=alter,
                     semester=semester)
```

Wenn die Variablen V1, V2 und V3 hätten heißen sollen:
`data.frame(V1=namen, V2=...)`.

- Wie groß ist der Datensatz?

```
dim(studis)  
  
# Zeilen und Spalten  
nrow(studis)  
ncol(studis)
```

- Rufen wir das Objekt `studis` auf, werden uns alle Beobachtungen angezeigt
- Nachdem die Variablen zu einem Datensatz zusammengefasst wurden, stellt sich die Frage, wie man an einzelne Variablen darin kommt.

- R kennt zwei Modi, um Variablen in einem Datensatz anzusprechen¹

```
# Allgemeine Form für Beobachtungen und Variablen  
studis[Zeilennummer(n), Spaltennummer(n)]
```

```
# Spezielle Form für Variablen  
studis$Spaltenname
```

- Die für uns wichtige Variante, ist die zweite Schreibweise

¹Der Spezialfall eines `data.frame()` kann verallgemeinert werden zu sämtlichen R Objekten.

- Variablen eines Datensatzes werden wir immer durch ein Dollarzeichen ansprechen
- Beispiel: Wenn wir das Alter der Studis anzeigen lassen möchten

```
# Variable alter im Datensatz studis ausgeben  
studis$alter
```

Die Dollarzeichen Variante mag, wenn man mit Stata vertraut ist, ungewohnt erscheinen. In Stata werden Variablen immer direkt angesprochen. Das liegt daran, dass man in Stata in der Regel nur mit einem Datenobjekt und einem Ergebnisobjekt arbeitet. Also ist `studis$` in Stata nicht notwendig, weil es ohnehin nur das Datenobjekt `studis` gibt.

- Einzelne Elemente wählen wir mit eckigen Klammern aus

```
# Die Angaben der ersten Beobachtung (Zeilen)
studis[1, ]

# Nur die Alter (Spalten)
studis[ , 2]
studis[ , "alter"]

# Namen und Alter (Spalten)
studis[ , c(1,2)]
studis[ , c("namen", "alter")]

# Das Alter der 3. Person (Zeilen und Spalten)
studis[3, 2]
studis[3, "alter"]
```

- Erneut wollen wir etwas über das Alter der Studis wissen

```
# Mittelwerte  
mean(studis$alter)  
median(studis$alter)  
  
# ausserdem min und max Alter  
min(studis$alter)  
max(studis$alter)
```

In Stata hieße ein Aufruf für den Datensatz studis z.B. mean alter oder für alles in einem tabstat alter, statistics(mean median min max).
In R brauchen wir Klammern um Befehle und müssen den Datensatz benennen, in dem die interessierende Variable vorliegt.

- Die Namen interessieren uns gar nicht so, dafür das Geschlecht

```
# Variable "Namen" loeschen  
studis$namen <- NULL  
  
# Variable "useR" einfuegen  
studis$user <- c(0, 1, 1)
```

- Welche Variablen waren jetzt noch im Datenframe?

```
names(studis)
```

- Verzeichnis wechseln

```
setwd("Z:/Daten")
```

- R-Objekte speichern

```
save(studis, file="studis.Rdata")
```

Vermeiden Sie wenn möglich Ordner- und Dateinamen mit Umlauten oder Leerzeichen.

- Verzeichnis anzeigen

```
dir()
```

- Daten laden

```
load("studis.Rdata")
```

- Mit `load` und `save` können Sie alle möglichen R-Objekte speichern und laden, nicht nur Datensätze.
- Rdata Objekte sind nicht gut geeignet zur Weitergabe, da bei Verwendung von unterschiedlichen R-Versionen und Betriebssystemen Probleme auftreten können.
- Es ist oftmals sinnvoll Daten direkt in einem Austauschformat zu speichern, etwa als CSV-Datei – später mehr dazu.

- Wenn wir wissen, wie ein Befehl heißt, aber nicht wissen was er macht, brauchen wir die Hilfe

```
# load() laedt Daten, aber welches Objekt braucht load()?  
load()
```

- Hilfe zum Befehl

```
help(load)  
# alternativ  
?load
```

Und wie kommt man an Befehle?

```
help.search("load")  
# alternativ  
??load
```

Und sonst?

- Google ist dein Freund!
- Etwas spezieller: rseek.org
- Alle R-Pakete: cran.r-project.org (auch nach Anwendungsbereich "Task")
- Geh zu einer R Nutzergruppe²

help.search kann Hinweise liefern, muss aber nicht. Suchen Sie in jedem Fall nach den englischen Begriffen.

²; -)

Syntax und ich

- Dateikopf: Dateiname, der Name des Erstellers, das Erstellungsdatum, eine kurze Information, was die Datei macht und bei längeren Dateien ein kurzes Inhaltsverzeichnis.

```
# Name: read-allbus.R  
# Autor: Garbuszus/Jeworutzki  
# Datum: 2016-10-26  
# Inhalt: Einlesen der ALLBUS-Daten
```

Dokumentieren Sie Ihre Syntax!

- Teamarbeit
- Vergesslichkeit

Niemand möchte sich Ihre Syntax erst mühsam erarbeiten.
Dokumentieren Sie viel und gründlich - Sie werden vergessen!

- Eine Raute (#) definiert den Beginn eines Kommentars.

```
# Das Ergebnis aus der Zeile darunter ist 2.  
1+1
```

- Befehle in Kommentaren werden nicht ausgeführt!

Verschiedene Aufgaben, verschiedene R-Files.

- Daten einlesen
- Aufbereitung
- Deskriptive Auswertungen
- usw.

Nicht jedes R-File wird immer gebraucht. Lesen Sie z. B. einmal die Daten ein, bereiten Sie sie einmal auf. Das muss nicht für jede neue Kreuztabelle erfolgen. Zeit ist Geld!

R-Files können ihrerseits auch aus R-Files aufgerufen werden

```
# R-Files aufrufen
source("DatenEinlesen.R")
source("Aufbereitung.R")
source("DeskriptiveAuswertungen.R")
# Ende
```

Dadurch wird die Syntax schlanker und die R-Files werden garantiert in der richtigen Reihenfolge aufgerufen.

Deskription

- Wieviel unterschiedliche Alter haben wir?

```
table(studi$alter)
```

- Mehr R User?

```
table(studis$userR)
```

- Und gekreuzt?

```
table(studis$userR, studis$alter)
```

- Und relative Häufigkeiten?

```
# Totalprozente
prop.table( table(studis$user, studis$alter) )

# Bei zwei Gruppen, haengen wir hier eine zwei an, um
# die Spaltenprozente bzw. -anteile zu ermitteln.
prop.table( table(studis$user, studis$alter), 2 )

# Zeilenprozente erhalten wir durch anhaengen von "1"
prop.table( table(studis$user, studis$alter), 1 )
```

Lassen sie ruhig Leerzeichen, um ihre Syntax übersichtlich zu gestalten. Als Option könnte man im Beispiel hier, die Tabelle erst in ein Objekt schreiben.

- Waren UseR älter?

```
# Studis die kein R nutzen sind durchschnittlich ...  
# Jahre alt.  
mean( studis$alter[studis$userR==0] )  
  
# und UseR sind durchschnittlich ... Jahre alt.  
mean( studis$alter[studis$userR==1] )
```

Was hier durch indizieren gemacht wird,
wird in Stata über die **if**-Funktion erzielt.

- Mittleres Alter der Personen mit mehr als 3 Semestern

```
mean( studis$alter[studis$semester > 3] )
```

Der Index (das was in den eckigen Klammern steht) ist eine logische Bedingung. Hier wird die Variable `alter` zurück gegeben, für Stellen, an denen der Index `TRUE` ergibt.

- Fehlende Werte in R werden als NA geschrieben, wir setzen das Alter der 3. Person auf NA

```
studis$alter[3] <- NA  
  
# anzeigen  
studis
```

- Unterschiedliche Befehle handeln die fehlenden Werte unterschiedlich

```
# gibt NA zurueck, wenn ein Wert NA ist
mean(studis$alter)

# gibt den Mittelwert zurueck
mean(studis$alter, na.rm=TRUE)

table(studis$alter)
# mit fehlenden Werten
table(studis$alter, useNA="ifany")
```

- Wir haben nun ausreichend lange mit dem Phantasiedatensatz der Studis rumgespielt
- Bitte einmal die Daten des ALLBUS einlesen

```
# Die Allbus Stata Daten wurden mit den untenstehenden  
# Befehlen eingelesen.  
# library("foreign")  
# dat <- read.dta("ZA4614_v1-1-1.dta",  
#               convert.factors=F)  
# save(dat, file="Allbus.Rdata")  
  
# Daten laden  
load("Allbus.Rdata")  
  
# Umfang des Datensatzes  
dim(dat)
```

- Einige uns interessierende Variablen

```
# Leeres Listen-Objekt erzeugen und  
# Variablen unter neuen Namen anfüegen  
dd <- NULL  
dd$sex      <- dat$v217  
dd$gebjahr  <- dat$v219  
dd$alter    <- dat$v220  
dd$ost      <- dat$v8  
dd$kgr      <- dat$v593 # koerpergroesse  
dd$kgw      <- dat$v595 # gewicht  
dd$inc      <- dat$v344  
  
# In Datensatz umwandeln und Zusammenfassung anzeigen  
dd <- data.frame(dd)  
summary(dd)
```

Der neue Datenframe ist nicht zwingend notwendig. Damit haben wir aber ein Objekt erzeugt, welches aus nur noch sieben Variablen besteht.

- Einige fehlende Werte müssen umkodiert werden

```
# Daten um "keine Angabe", "weiss nicht" bereinigen.  
# 9999 "keine Angabe"  
dd$gebjahr[dd$gebjahr==9999] <- NA  
# 999 "keine Angabe" (aus v219)  
dd$alter[dd$alter==999] <- NA  
  
# 996 "keine Teilnahme"  
# 998 "weiss nicht"  
# 999 "keine Angabe"  
dd$kgr[dd$kgr>=996] <- NA  
dd$kgw[dd$kgw>=996] <- NA  
  
# 99997 "Angabe verweigert"; 99999 "keine Angabe"  
dd$inc[dd$inc>=99997] <- NA
```

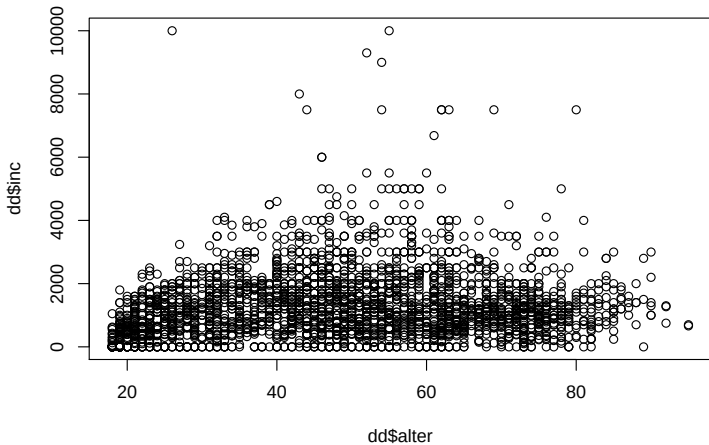
- Ein erstes Streudiagramm vermittelt noch eine Reihe von Ausreißern

```
# Alter und Einkommen
plot(x=dd$alter, y=dd$inc)

# Auf geringe Einkommen einschraenken
plot(x=dd$alter[dd$inc<15000], y=dd$inc[dd$inc<15000])

dd <- subset(dd, dd$inc<15000)
plot(x=dd$alter,y=dd$inc)
```

Allbus: Der Plot

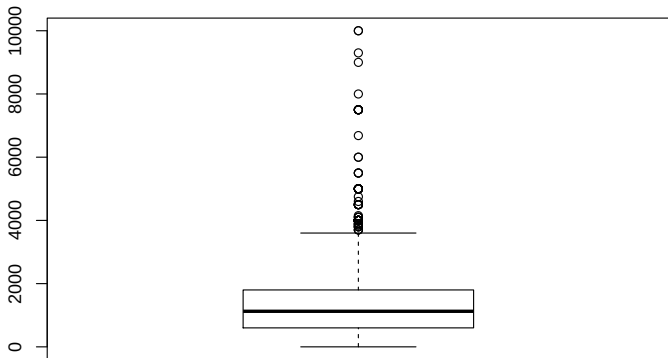


- Wie in der Statistik VL gelernt, der Boxplot

```
boxplot(dd$inc)

# die Statistiken dazu
quantile(dd$inc, na.rm=TRUE)
min(dd$inc, na.rm=TRUE)
max(dd$inc, na.rm=TRUE)
median(dd$inc, na.rm=TRUE)
```

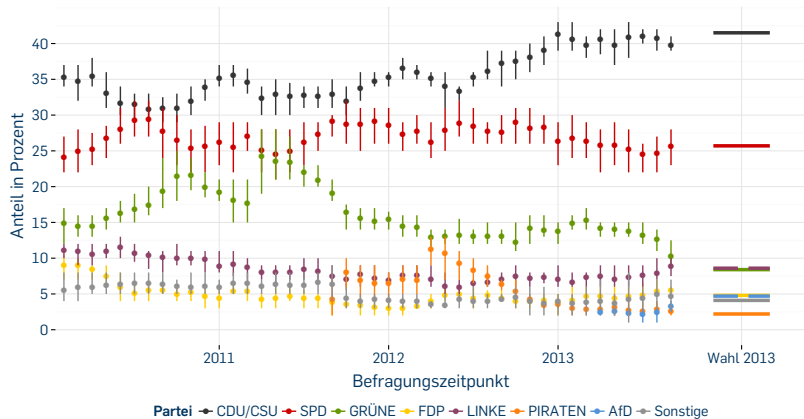
Allbus: Der Boxplot

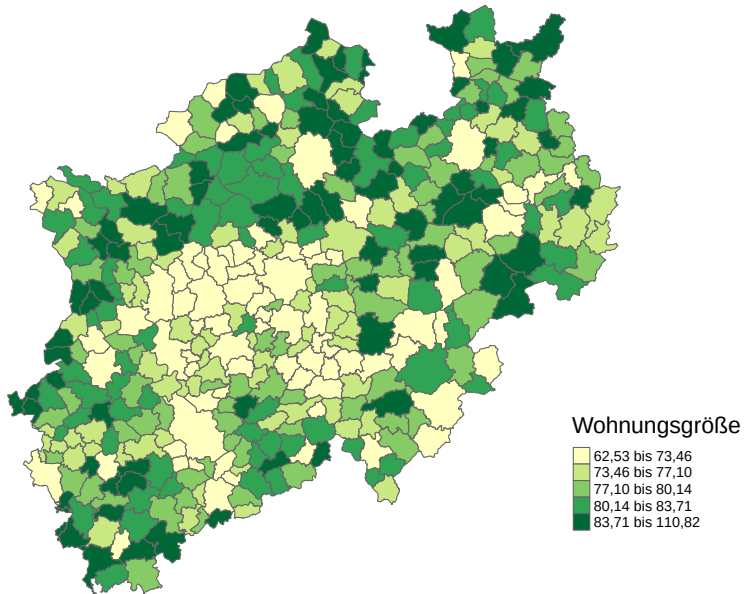


- Weitere Grafiken

```
# Histogramm
hist(dd$inc)
# Dichtekurven
plot(density(dd$inc))
# Dotchart
dotchart( c(mean(dd$inc[dd$sex==1]),
             mean(dd$inc[dd$sex==2])),
          c("m", "w"))
```

- Grafiken können zudem noch mit diversen Parametern hübscher gestaltet werden.
- Überschriften, Achsenbeschriftungen, Symbole, Legenden





Multivariate Statistik

- Wir hatten alter und inc bereits im Scatterplot betrachtet
- Wie kann man den Zusammenhang besser beschreiben? Mit einer Regression!

```
# lineare Regression rechnen und Ergebnisse abspeichern
erg1 <- lm(inc~alter, data=dd)
erg1

# Tabelle mit den Reg.-Koeffizienten und Teststatistiken
summary(erg1)
```

Die erste Regression

```
> summary(erg1)
```

Call:

```
lm(formula = inc ~ alter, data = dd)
```

Residuals:

Min	1Q	Median	3Q	Max
-1621.4	-714.1	-229.4	500.2	8857.5

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	944.862	58.174	16.242	< 2e-16 ***
alter	7.601	1.115	6.819	1.12e-11 ***

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

Residual standard error: 1062 on 2776 degrees of freedom
(1 observation deleted due to missingness)

Multiple R-squared: 0.01648, Adjusted R-squared: 0.01612

F-statistic: 46.5 on 1 and 2776 DF, p-value: 1.118e-11

- Die Regressionsgerade können wir grafisch darstellen

```
plot(x=dd$alter, y=dd$inc)

# abline uebernimmt die Werte für Steigung und
# Achsenabschnitt
abline(erg1, col="red")
```

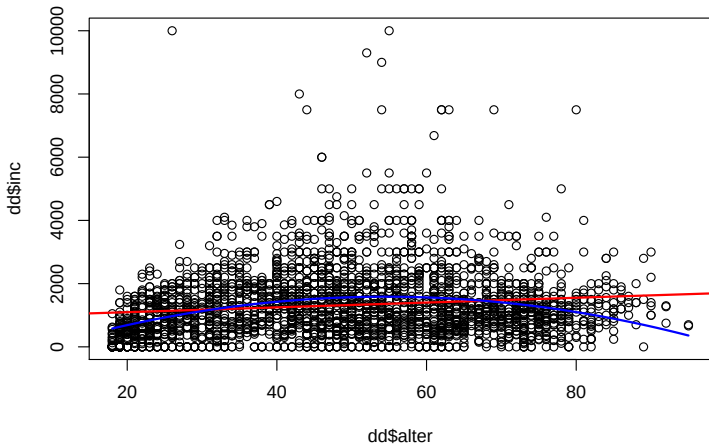
- Das Alter sollte hier nicht-linear eingehen

```
erg2 <- lm(inc~alter+I(alter^2), data=dd)
erg2
summary(erg2)
```

- Zudem können wir die geschätzten Einkommen berechnen und einzeichnen

```
p1 <- predict(erg2, newdata=data.frame(alter=18:95))  
lines(x=18:95, p1, col="blue", lwd=3)
```

Regressionsgerade



Material zum Selbststudium



Einführung in die Statistik mit R
Andreas Behr und Ulrich Pötter



Programmieren mit R
Uwe Ligges



The R Book
Michael Crawley



Sozialwissenschaftliche Datenanalyse mit R
Katharina Manderscheid



Reihe: Use R!
Springer Verlag

R-Skript	Umfassende Einführung von Dudel/Jeworutzki
Datacamp	Grundlegende Befehle und Datenstrukturen
O'Reily	Grundlegende Befehle
Codeschool	

- Offizielle Seite zu R
- Planet R: Sammlung von Blogposts zu R
- R-Bloggers: Sammlung von Blogposts zu R
- Suchmaschine zu R Fragen
- Journal of Statistical Software
- Quick-R: Kompakte Tipps und Anleitungen zu R
- R-Forum bei stackoverflow

Vielen Dank für die Aufmerksamkeit!

Und noch viel mehr ...

- R ist eine Turing-vollständig Programmiersprache
- Daher können wir selbst umfangreiche Programme schreiben und dabei die grundlegenden Werkzeuge einer Programmiersprache, Funktionen und Kontrollstrukturen, nutzen.
- Wir besprechen
 - Funktionen `function(x){...}`
 - Bedingungen `if (x){...}else {...}`
 - Schleifen `for (i in 1:10){...}`

- Alle wiederkehrenden Abläufe sollten in Form von wiederverwendbaren Funktionen geschrieben werden.
- Generell kann das Schreiben von Funktionen die Übersichtlichkeit erhöhen.

Funktionen

```
# Funktion um den Kehrwert zu berechnen
kehrwert <- function(x) {
  z <- 1/x
  return(z)
}

kehrwert(10)
```

- Mit dem Befehl `function(x)` wird eine Funktion erzeugt und der auszuführende R-Code steht in den geschweiften Klammern
- `x` ist ein Parameter der Funktion, der beim Aufruf angegeben werden muss und innerhalb der geschweiften Klammern genutzt werden kann.
- `return(z)` sorgt dafür, dass das Ergebnis `z` zurückgegeben wird

Funktionen

- Ein etwas aufwändigeres Beispiel zur Berechnung von Cramer's V
- Dabei wird ein neuer Befehl verwendet `if`

```
# Cramers V
cV <- function(x){
  if (is.matrix(x)==T){
    return(
      sqrt(
        as.numeric(chisq.test(x)$statistic) /
        sum(x) * min(dim(x) -1)
      )
    )
  } else {
    print("x is not a matrix")
  }
}
```

- Um die Ausführung von Bedingungen abhängig zu machen verwendet man `if (BEDINGUNG)`
- Die Bedingung muss ein logischer Ausdruck sein, der die Ausprägungen TRUE oder FALSE annehmen kann

- Wenn das Objekt in der Klammer den Wert TRUE annimmt, wird der nachfolgende Befehl ausgeführt

```
if (1 < 2) { print("1 ist kleiner als 2") }
```

- Eine if Bedingung kann allerdings auch negierend genutzt werden (z. B. in Schleifen)

```
if (!1 > 2) { print("Stimmt") }
```

- if-Bedingungen können zudem verschachtelt werden.

- `if` ist u.a. nützlich, um auf unterschiedliche Eingabewerte für eine Funktion zu reagieren
- Mit `else` kann eine Aktion vorgegeben werden, wenn die Bedingung für `if` `FALSE` ist.
- Über `if (x)...else if (y)...else ...` können mehrere Bedingungen nacheinander geprüft werden

if – ein Beispiel

Eine Funktion die **if** und **else** verwendet:

```
# Wenn x metrisch ist, den Mittelwert berechnen,  
# ansonsten eine Häufigkeitstabelle  
deskription <- function(x) {  
  if (is.numeric(x)) {  
    return(mean(x))  
  } else {  
    return(table(x))  
  }  
}
```

For-Schleifen

- Mit den Schleifen können wir uns Arbeit erleichtern und Aufgaben für eine Vielzahl von Werten wiederholt ausführen
- Im Aufruf `for (i in 1:10){...}` wird der Inhalt der Schleife 10 mal ausgeführt, im ersten Durchlauf ist `i=1`, im zweiten `i=2` usw.

```
for (i in 1:10) {  
    print(i)  
}
```

While-Schleifen

- `while`-Schleifen dienen dazu, Aufgaben solange zu wiederholen bis eine Bedingung nicht mehr erfüllt ist
- Im Beispiel gibt die Funktion solange Zufallswerte aus bis ein Wert größer 0.8 erzeugt wurde

```
x <- 0.5
while (x < 0.8) {
  x <- rnorm(1)
  message(x)
}
x
```

While-Schleifen: break

- while-Schleifen können zusätzlich mit dem Befehl `break` unterbrochen werden
- Im Beispiel wird die Schleife auch dann abgebrochen, wenn ein Wert kleiner als -1 erzeugt wurde

```
x <- 0.5
while (x < 0.8) {
  x <- rnorm(1)
  message(x)
  if (x < -1) {
    break
  }
}
x
```

- Wenn Berechnungen in einer Schleife in einem Objekt gespeichert werden sollen (`x[i] <- ...`), sollte die Größe der Objekte wenn möglich schon vorab festgelegt werden (`x <- rep(NA, 100)`). Dies beschleunigt die Ausführung und spart Arbeitsspeicher.
- Teilweise ist es eleganter auf die alternativen Funktionen `apply`, `lapply` usw. zurückzugreifen.

Alternativen für For-Schleifen

Varianten von apply

- | | |
|------------------------------------|--|
| <code>apply(x, 1, function)</code> | Schleife über die Zeilen (1) oder Spalten (2) einer Matrix oder eines Arrays |
| <code>lapply(x, function)</code> | Schleife über die Elemente einer Liste oder Vektors/die Spalten eines Dataframes. Erzeugt eine Liste. |
| <code>sapply(x, function)</code> | Schleife über die Elemente einer Liste, versucht das Ergebnis zu vereinfachen (z.B. als Vektor oder Dataframe) |
| <code>tapply(x, function)</code> | Schleife über Untergruppen eines Vektors, die durch einen Faktor definiert werden |

apply

- Mit `apply` kann man über die Zeilen oder Spalten einer Matrix iterieren.

```
data(iris)
head(iris)

# Wir wollen nur numerische Spalten
iris.num <- iris[,1:4]

dim(iris.num)

# Zeilen: Durchschnitt ueber alle Zeilenwerte
apply(iris.num, 1, mean)

# Spalten: Durchschnitt ueber alle Spalten
apply(iris.num, 2, mean)
```

- Mit `lapply` iteriert man über die Elemente einer Liste oder die Spalten eines Dataframes. Als Ergebnis erhält man eine Liste zurück.

```
data(iris)
head(iris)
iris.num <- iris[,1:4]

# Liste mit den Quantilen fuer alle Variablen in iris.num
lapply(iris.num, quantile)
```

- sapply vereinfacht das Rückgabeobjekt, falls möglich
- Ist das Ergebnis für jede Iteration ein einzelner Wert, wird ein Vektor erzeugt
- Sind die Ergebnisse für alle Iterationen gleich lange Vektoren, wird eine Matrix erzeugt

```
data(iris)
iris.num <- iris[,1:4]

# Matrix mit den Quantilen
sapply(iris.num, quantile)
```

- Mit `tapply` wird nicht über einzelne Elemente iteriert, sondern über Untergruppen, die ein Faktor definiert
- So kann bspw. ein bedingter Mittelwert für die Variable `Sepal.Length` getrennt für alle Spezies berechnet werden
- Alternativ kann der Befehl `by` genutzt werden, der anstelle eines Vektors eine Liste erzeugt

```
data(iris)

# tapply berechnet den Mittelwert getrennt fuer
# jede Spezies aus
tapply(iris$Sepal.Length, iris$Species, mean)
```

apply und anonyme Funktionen

- Es können auch komplexe Funktionen mit `apply` und `co` berechnet werden
- Eine kompakte Möglichkeit ist die Verwendung von anonymen Funktionen. Diese werden als `anonym` bezeichnet, weil sie keinen Namen haben und nur innerhalb des `sapply`-Aufrufs bestehen.

```
data(iris)

# Die Funktion berechnet nur dann den Durchschnitt,
# wenn x numerisch ist ansonsten ist das Ergebnis -1
sapply(iris, function(x) {if (is.numeric(x)) mean(x) else -1})
```

- Für viele Anwendungen wurde **R** um Funktionen erweitert, die in Bibliotheken (auch Pakete genannt) zur Verfügung gestellt werden.
- Pakete lassen sich mit einem Funktionsaufruf installieren.

```
install.packages("paketname")
```

- und dann zur Benutzung laden:

```
library("foreign")  
library("Rcpp")  
library("SemiPar")  
library("dplyr")  
library("MASS")  
library("ggplot2")  
library("Hmisc")  
library("knitr")
```