Einführung in R - Eine Umgebung für statistisches Rechnen

Genomische Datenanalyse

2.Kapitel

Übersicht

- Einleitung
- Ausdrücke
- Daten und Datentypen
- Kontrollstrukturen
- Illustration und Grafik

Was ist R?

- Interaktive Umgebung für statistisches Rechnen
- Bedienung über Kommando-Zeile
- Objekt-orientierte Sprache
- Aufwendige Möglichkeiten zur Erweiterung

Entstehung von R

- Entwicklung der Sprache S an Universitäten
- Kommerzialisierung von S und Splus als professionelle Statistikpakete
- Abspaltung von R im akademischen Bereich (weitgehend S-kompatibel)

Verfügbarkeit & Installation

- http://www.r-project.org
- Source code unentgeltlich verfügbar
- Test und Entwicklung für einige Platformen: Linux, Windows, MacOS
- Teilweise auch ausführbare Programme zur einfachen Installation verfügbar

Resourcen & Hilfe

- Wie immer die wichtigsten Kommandos:
  - `help.start()`
  - `help.search("Begriff")`
  - `help(Funktionsname)`
- FAQs, Mailing-Listen, Tutorials – alles umsonst vom r-project
- Ach ja, auch sehr wichtig: q()

Einfache Ausdrücke

- **Mit Zahlen:**
  - Arithmetik: +, -, *, /, ^, %%, %/%
  - Trigonometrie: sin, cos, tan, asin, acos,...
  - Weitere Funktionen: abs, sign, log, exp, sqrt, ...
  - Logische Operatoren: <, >, ==, !=, !, |, &, ...
- **Mit Zeichenketten:**
  - Schneiden & Kleben: paste, substr, strsplit...
  - Formatieren: sprintf, format,...
  - Finden & Ersetzen: grep, sub, match,...

Erstellen von Vektoren

- Erstellen: c, rep, seq, :

```
> c(1, 2, 3)    > rep(2, 3)
[1] 1 2 3       [1] 2 2 2
> 1:3           > seq(1, 1.4, 0.2)
[1] 1 2 3       [1] 1.0 1.2 1.4
```

Verketten: c

```
> c(seq(3), 4:6)
[1] 1 2 3 4 5 6
```

Ausdrücke mit Vektoren

- **Elementweise Arithmetik und Funktionen:**

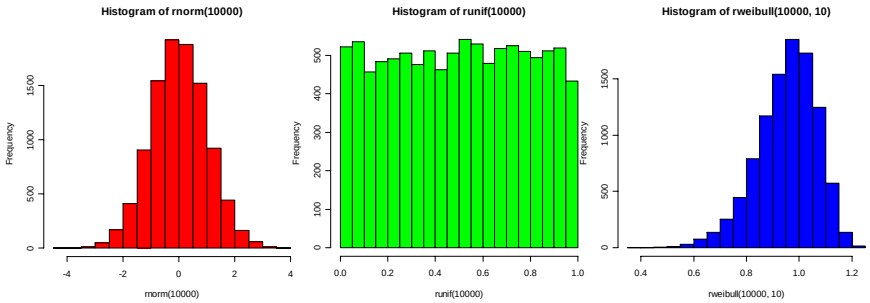
```
> sqrt(c(1,9,25))
[1] 1 3 5
> c(1,2,3)*c(2,3,4)
[1] 2 6 12
```

- **Duplikate entfernen: unique**

```
> unique(c(1, 3, 3, 4, 1))
[1] 1 3 4
```

Zufallszahlen Generieren

- **Aus verschiedenen Verteilungen: rnorm, runif, rhyper,...**



Eigenschaften von Vektoren

- **Eigenschaften: sum, prod, mean, median, min, max, quantile,,...**

```
> sum(rnorm(1000))
[1] 23.22438
```

- **Alles in einem: summary**

```
> summary(rnorm(1000))
 Min. 1st Qu. Median Mean 3rd Qu.  Max.
-3.25 -0.7378 -0.043 -0.0 0.65120 3.452
```

Statistik mit Vektoren

- **Weitere Kenngrössen: var, cov, cor,...**

```
> var(rnorm(1000))
[1] 1.009406
```

- **Ordnung schaffen: sort, order, rank**

```
> sort(c(4,2,3,1))
[1] 1 2 3 4
> order(c(4,2,3,1))
[1] 4 2 3 1
```

Ausdrücke mit Mengen

- Ein Vektor kann als Menge von Elementen aufgefasst werden
- Test auf Existenz: %in%, is.element

```
> c(1,2,3) %in% c(1,3,4)
[1] TRUE FALSE TRUE
```

- Algebra: setdiff, union, intersect

```
> setdiff(c("a","b","c"), c("b","d"))
[1] "a" "c"
```

Erstellen von Matrizen

- Eine Matrix ordnet Elemente in Zeilen und Spalten an

```
> matrix(rnorm(8), nrow=2)
        [,1]   [,2]   [,3]    [,4]
[1,] -0.847 -0.436 -0.927  0.6134
[2,] -0.158  0.896 -1.378  0.2674
```

- Kombinieren von Vektoren oder Matrizen: cbind, rbind

Ausdrücke mit Matrizen

- Elementweise Arithmetik
- Häufigkeitstabellen: table

```
> table(c(1,1,2,1,2,2,1),
        c(2,2,1,1,2,1,2))
  1 2
1 1 3
2 2 1
```

Variablen und Zuweisungen

- Zahlen und Zeichenketten werden in Variablen gespeichert:

```
> x <- 1.23
> y <- „Hello World“
```

- Gross-/Kleinschreibung wird bei Variablennamen berücksichtigt.
- Variablen verwalten: ls, rm

Einfache Datentypen

- **Variablen können Werte der folgenden Typen speichern:**
  - Zahlen (numeric)
  - Zeichenketten (character)
  - Boolsche Werte (logical, TRUE/FALSE)
  - Komplexe Zahlen (complex)

Komplexe Datentypen: Vektoren erstellen

- **Erstellen:**

```
> x <- c(1,2,3)        > length(x)
[1] 1 2 3              [1] 3
```

- **Elemente benennen:**

```
> names(x) <- c("a", "b", "c")
> x
a b c
1 2 3
```

Komplexe Datentypen: Auf Vektoren zugreifen

- **Auf Elemente zugreifen:**

```
> x[1]          > x["b"]
[1] 1           [1] 2
```

- **Elemente verändern:**

```
> x["c"] <- 5
```

„Alle ausser“: negative Indizes

```
> x[-2]
a c
1 5
```

Komplexe Datentypen: Vektoren benutzen

- **Mit boolschen Vektoren auf Elemente zugreifen:**

```
> x[c(TRUE,TRUE,FALSE)]
a b
1 2
```

Mit logischen Ausdrücken auf Elemente zugreifen:

```
> x > 1
[1] FALSE TRUE TRUE
> x[x > 1]
b c
2 5
```

Komplexe Datentypen: Matrizen erstellen

- Erstellen von Matrizen

```
> m <- matrix(1:6, ncol=3)
```

- Benennen von Zeilen und Spalten

```
> colnames(m) <- letters[1:3]
> rownames(m) <- letters[4:5]
  a b c
d 1 3 5
e 2 4 6
```

Komplexe Datentypen: Auf Matrizen zugreifen

- Auf Zeilen und Spalten wird analog zu Elementen von Vektoren zugegriffen

```
> m[1,1]    > m[“d“,]
[1] 1       a b c
            1 3 5
```

- Eigenschaften von Matrizen: dim, ncol,nrow, …

```
> dim(m)
[1] 2 3
```

Komplexe Datentypen: Matrizenmultiplikation

- In R direkt definiert

```
> n <- m  %*% t(m)

  a b c          d e
d 1 3 5  %*%  a 1 2         d  e
e 2 4 6       b 3 4  =  d 35 44
              c 5 6     e 44 56

n[“d“, “e“] <- 1*2 + 3*4 + 5*6
```

Umwandlung (Coercion)

- Einfache Datentypen: as.numeric, as.character, as.logical

```
> as.numeric(“1.2“)
> as.logical(“T“)
```

- Komplexe Datentypen: as.vector, as.list, as.matrix

```
> as.matrix(c(1,2,3))
```

Dataframes benutzen

- Interaktiv erstellen (Spreadsheet):
  ```
  > new.df <- edit(data.frame())
  ```
- Aus Vektoren zusammensetzen:
  ```
  > new.df <- data.frame(x=c(1,2,3),
                         y=c(4,5,6))
  ```
- Zeilen und Spaltennamen müssen eindeutig sein: row.names, col.names
- Zugriffsvarianten:
  ```
  new.df[,2], new.df[,"y"], new.df$y
  ```

Mitgelieferte Datensätze

- R enthält einige Beispieldatensätze, eine Übersicht erhält man mit:
  ```
  > data()
  ```
- Der Datensatz "USArrests" aus dieser Übersicht wird wie folgt geladen:
  ```
  > data(USArrests)
  ```
- Auch über Datensätze gibt's Hilfe:
  ```
  > help(USArrests)
  ```

Dataframe Eigenschaften

- Grösse des Datensatzes: dim, str
- Überblick der Inhalte: summary

```
> summary(USArrests)
```

```
     Murder          Assault         UrbanPop          Rape
 Min.   : 0.800   Min.   : 45.0   Min.   :32.00   Min.   : 7.30
 1st Qu.: 4.075   1st Qu.:109.0   1st Qu.:54.50   1st Qu.:15.07
 Median : 7.250   Median :159.0   Median :66.00   Median :20.10
 Mean   : 7.788   Mean   :170.8   Mean   :65.54   Mean   :21.23
 3rd Qu.:11.250   3rd Qu.:249.0   3rd Qu.:77.75   3rd Qu.:26.18
 Max.   :17.400   Max.   :337.0   Max.   :91.00   Max.   :46.00
```

Import von Daten

- Aus R abgespeicherte Daten: load
  ```
  > load("myfile.rda")
  ```
- Daten aus anderen Quellen, z.B. als Tabelle in der Elemente mit Tabulatoren getrennt sind
  ```
  > read.table("myfile.txt", sep="\t")
  ```

Tabellen als Textdatei

- Zeile mit Spaltennamen gefolgt von 7 Datenpunkten

```
Sepal.Length    Sepal.Width Species
5.1       3.5   setosa
4.9       3.0   setosa
4.7       3.2   setosa
7.0       3.2   versicolor
6.4       3.2   versicolor
6.9       3.1   versicolor
6.3       3.3   virginica
5.8       2.7   virginica
7.1       3.0   virginica
```

Export von Daten

- Verzeichnisse: getwd, setwd, dir
- R Variablen speichern: save
- Eine gesamte Session speichern: save.image
- Tabellen für andere Programme speichern: write.table
- Protokoll speichern: savehistory

Bedingte Ausführung

- Bedingung: if

```
>
> if (Murder[i]/Assault[5] > 0.1) {
+   cat(“high risk in sate“, i, “\n“)
+ } else {
+   cat(“less risk in sate“, i, “\n“)
+ }
```

- Verzweigung: switch

```
> switch(“cc“, a=1, cc=2, d=3)
[1] 2
```

Iterierte Ausführung

- Schleifen: for, while, repeat

```
> for (i in c(“a“, “cc“, “d“)) print(i)
[1] “a“
[1] “cc“
[1] “d“
```

- Zeilen-/Spaltenschleifen: apply

```
> apply(m, 2, mean)
[1] 1.5 3.5 5.5
```

Funktionen

- Funktionen werden Variablen zugeordnet
- Sie enthalten formale Argumente und einen Befehlsblock
- Resultate zurückgeben: return

Beispiel einer Funktion

- Definition:

```
> computeArea <- function(r) {
+   return(pi*r^2)
+ }
```

- Aufruf:

```
> computeArea(2)
[1] 12.56637
```

Rekursive Funktionen

- Funktionen dürfen sich selbst aufrufen:

```
> fibo <- function(n) {
+   if (n < 3) return(1)
+   else return(fibo(n-2) + fibo (n-1))
+ }
```

- Aufruf:

```
> sapply(1:10, fibo)
[1] 1 1 2 3 5 8 13 21 34 45
```

Skripten schreiben

- Funktionen möchte man in einem Texteditor schreiben: edit

```
> fibo <- edit()
```

- Um Abfolgen von Befehlen zu bearbeiten, als Text speichern und aus R aufrufen

```
> source("myscript.R")
```

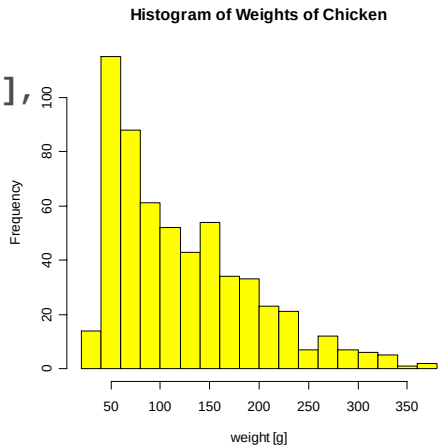
- Spezielle Editoren: RWinEdt, Emacs/ESS

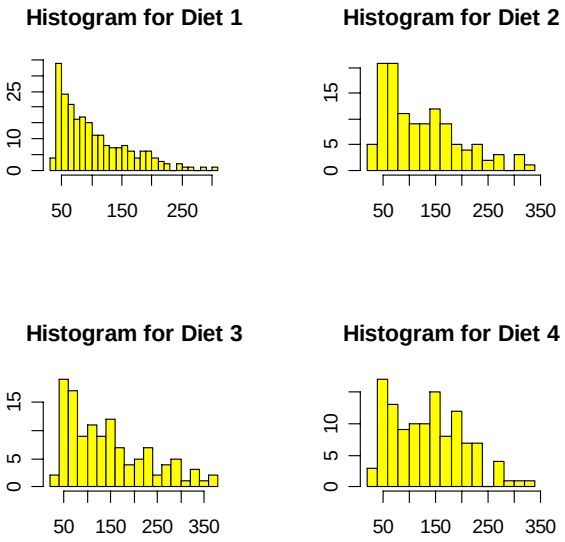
Grafiken in R

- Intuitives Mittel zur Analyse
- High-Level Funktionen für neue Grafiken: barplot, matplot, dotchart, contour, persp, ...
- Low-Level Funktionen zum Ergänzen von Grafiken: points, text, polygon, segments, arrows, rug, ...

Histogramme

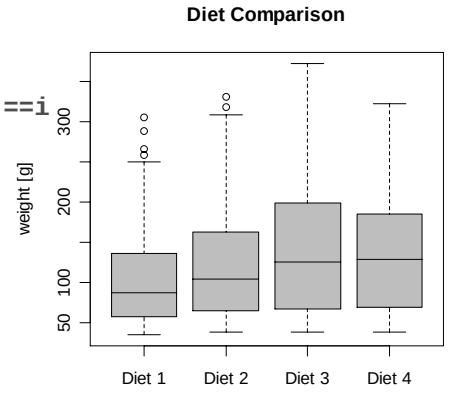
```
> data(ChickWeight)
> hist(ChickWeight[,1],
       main="...",
       xlab="...",
       col="yellow")
```





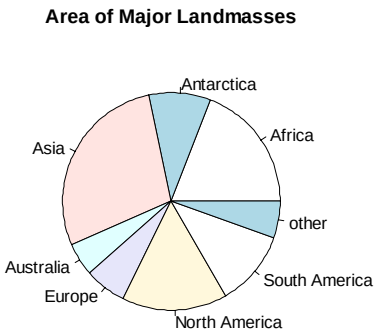
Boxplots

```
> perDiet <- list()
> for (i in 1:4) {
+   s <- ChickWeight[,4]==i
+   perDiet[[i]] <-
+     ChickWeight[s,1]
+ }
> boxplot(perDiet)
```



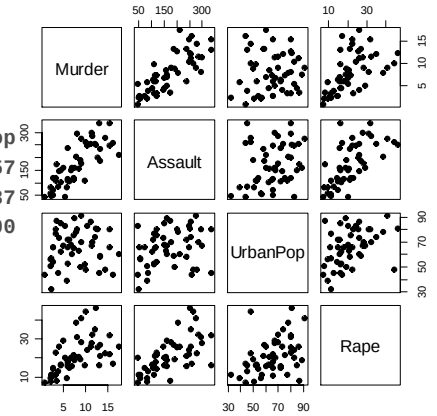
Kuchendiagramme

```
> data(islands)
> s <- which(islands>1000)
> nyIslands <- islands[s]
> other <- sum(islands[-s])
> myIslands <- c(myIslands,
                 other=other)
> pie(myIslands,main="...")
```



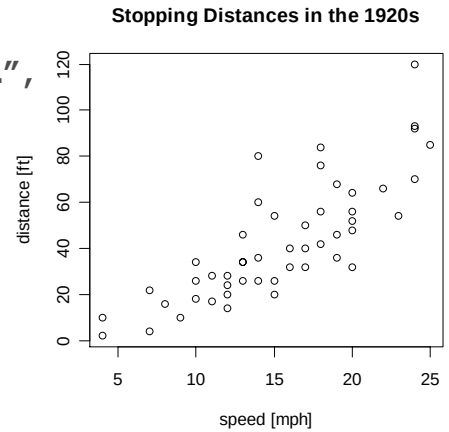
Streudiagramme

```
> data(USArrests)
> plot(USArrests)
> cor(USArrests[,1:3])
          Murder Assault UrbanPop
Murder   1.00000  0.8019  0.06957
Assault  0.80187  1.0000  0.25887
UrbanPop 0.06957  0.2589  1.00000
```



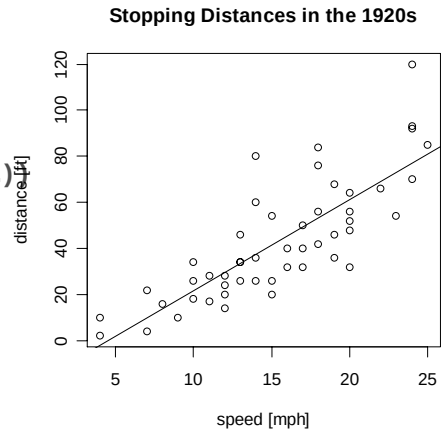
Abhängigkeit modellieren

```
> data(cars)
> plot(cars, main="…",
+      xlab="…",
+      ylab="…")
```



Abhängigkeit modellieren

```
> data(cars)
> plot(cars, main="…",
+      xlab="…",
+      ylab="…")
> abline(lm(dist~speed,…))
> cor(cars[,1],cars[,2])
[1] 0.807
```



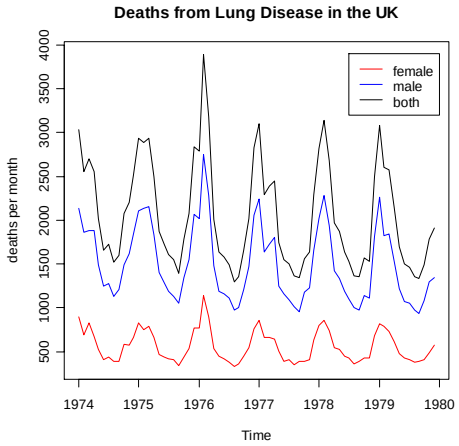
Zeitreihen

```
> data(UKLungDeaths)
> ls()
[1] fdeaths, ldeaths
[3] mdeaths
> print(fdeaths)
     Jan  Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1974 901  689 827 677 522 406 441 393 387 582 578 666
1975 830  752 785 664 467 438 421 412 343 440 531 771
1976 767 1141 896 532 447 420 376 330 357 445 546 764
1977 862  660 663 643 502 392 411 348 387 385 411 638
1978 796  853 737 546 530 446 431 362 387 430 425 679
1979 821  785 727 612 478 429 405 379 393 411 487 574
```

```
> class(fdeaths)
[1] ts
> frequency(fdeaths)
[1] 12
```

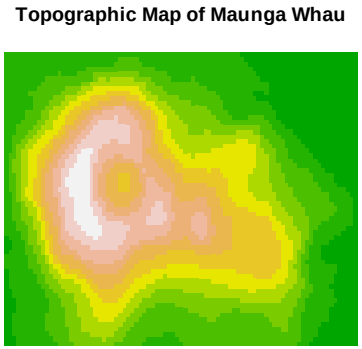
Liniendiagramme von Zeitreihen

```
> plot(ldeaths,
       main="...",
       ylab="...",
       type="l")
> lines(fdeaths),
        Col="red")
> lines(mdeaths),
        Col="blue")
> legend(...)
```



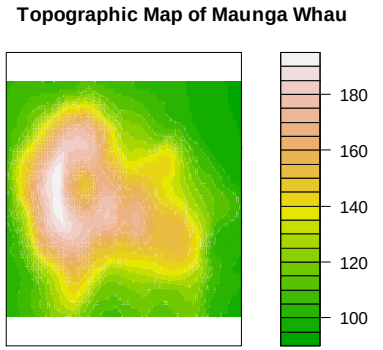
Images

```
> data(volcano)
> t <-terrain.colors(12)
> image(volcano,
+       col=t,
+       main="…")
```



Geglättetes Image

```
> data(volcano)
> t <-terrain.colors(12)
➢ filled.countour(
+           volcano,
+           color=t)
> title(main="…")
```



3D Image

```
> library(lattice)
> wireframe(volcano,
+     shade=TRUE,
+     aspect=c(61/87,0.4),
+     light.source = c(…))
> title(main="…")
```