LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Statistische Software

Micha Schneider

Institut für Statistik
Ludwig-Maximilians-Universität München

WS 2016/17, R Teil 2

Datentypen und -strukturen

Datentypen in R

Das einfachste Datenobjekt ist ein Vektor mit Elementen des Typs

- **numeric**: ganzzahlige oder Gleitkomma-Werte,
- **character**: beliebige Zeichen,
- **logical**: die Zustände `TRUE` und `FALSE`,
- **list**: ein Objekt beliebigen Typs, auch wieder eine Liste (rekursive Datenstrukturen!).

Jeder Vektor hat eine Länge (`length()`) und der Typ kann mittels `mode()` festgestellt werden.

```
> a <- 25
> mode(a)
[1] "numeric"
> length(a)
[1] 1
```


Datenstrukturen in R

Unter Datenstrukturen versteht man eine Beschreibung dessen, wie die Daten dargestellt werden und angeordnet sind:

- **Vektor**: Anordnung gleichartiger Elemente (z.B. nur Zahlen, nur characters, etc.)
- **Faktor**: spezielle Anordnung für kategoriale Daten
- **Matrix**: Anordnung gleichartiger Elemente in Zeilen und Spalten
- **dataframe**: Anordnung unterschiedlicher Elemente gleicher Länge
- **Liste**: Anordnung von verschiedenen Elementen (keine Vorgaben bzgl. Typ, Länge, etc.)

Numerische Vektoren

Üblicherweise wird `c()` (concatenate: verbinden) zum Erstellen von mehrelementigen Vektoren verwendet. Weitere wichtige Funktionen sind `seq()` oder `:` (Sequenzen) und `rep()` (Repeat).

```
> v1 <- c(1, 3.14, 17)
> v1
[1]  1.00  3.14 17.00
> v2 <- seq(from = -pi, to = pi, length = 5)
> v2
[1] -3.142 -1.571  0.000  1.571  3.142
> v3 <- 1:5
> v3
[1] 1 2 3 4 5
> v4 <- rep(2, 5)
> v4
[1] 2 2 2 2 2
```


Zeichen-Vektoren

Die Elemente von Zeichen-Vektoren bestehen aus einer Folge von beliebigen Zeichen. Zur Unterscheidung von Variablen werden Zeichenketten durch einfache oder doppelte Hochkomma am Anfang und Ende markiert.

```
> Benutzer <- c("Hansi", "Seppl")
> Benutzer
[1] "Hansi" "Seppl"
> mode(Benutzer)
[1] "character"
```


Logische Vektoren

Es gibt die zwei logischen Zustände `TRUE` und `FALSE`. In den meisten Fällen werden logische Vektoren nicht direkt eingegeben, sondern durch die logischen Operatoren ==, !=, . . . erzeugt. Der Operator ! invertiert einen logischen Vektor.

```
> WoIstSeppl <- (Benutzer == "Seppl")
> WoIstSeppl
[1] FALSE  TRUE
> ! WoIstSeppl
[1]  TRUE FALSE
> which(WoIstSeppl)
[1] 2
```


Zugriff auf Vektorelemente

Generell erfolgt in R der Zugriff auf Teile einer Struktur durch eckige Klammern. Bei einem Vektor können Zahlen und logische Vektoren für den Zugriff auf einzelne Elemente verwendet werden, negative Zahlen schließen einzelne Elemente aus.

```
> v2
[1] -3.142 -1.571  0.000  1.571  3.142
> v2[5]
[1] 3.142
> v2[2:4]
[1] -1.571  0.000  1.571
> v2[-(2:4)]
[1] -3.142  3.142
> v2[v2<0]
[1] -3.142 -1.571
```

Namen für Vektorelemente

Jedes Vektorelement kann einen Namen bekommen, diese können dann für den einfacheren Zugriff verwendet werden.

```
> konst <- c(e = exp(1), pi = pi, zweipi = 2 * pi)
> konst
     e     pi zweipi
 2.718  3.142  6.283
> names(konst)
[1] "e"      "pi"     "zweipi"
> names(konst)[3] <- "2pi"
> konst
    e    pi   2pi
2.718 3.142 6.283
> konst["2pi"]
  2pi
6.283
> konst[c("pi","2pi")]
   pi   2pi
3.142 6.283
```


Faktoren

Nominale oder ordinale Daten werden in R „Faktoren“ genannt. Intern ist dies ein ganzzahliger Vektor, wo jeder Zahl ein „Label“ zugeordnet ist. Mit der Funktion `factor()` lassen sich entsprechende Vektoren leicht erstellen.

```
> geschlecht <- c(0,1,1,0,0)
> geschlecht
[1] 0 1 1 0 0
> summary(geschlecht)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0     0.0     0.0     0.4     1.0     1.0
> geschlecht <- factor(geschlecht, levels=c(0,1), labels=c("maennlich","weiblich"))
> geschlecht
[1] maennlich weiblich  weiblich  maennlich maennlich
Levels: maennlich weiblich
> summary(geschlecht)
maennlich  weiblich
        3         2
```


Matrizen

Matrizen werden mit der Funktion
`matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE)`
erzeugt, wobei die Argumente folgende Bedeutung haben:

- data: Ein Vektor mit den Daten
- nrow: Anzahl der Zeilen
- ncol: Anzahl der Spalten
- byrow: Bei TRUE wird die Matrix zeilenweise aufgebaut, sonst spaltenweise

```
> X <- matrix(c(1,2,3,4,5,6),nrow=3)
> X
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```


Indizierung von Matrizen

Die Indizierung von Matrizen ist ähnlich wie der Zugriff auf Vektorelemente. Um den Wert mit Index `(i ; j)` also (Zeile i , Spalte j) einer Matrix X anzusprechen, verwendet man die Form: `X[i, j]`.
Das Weglassen einer Spaltenangabe, also etwa `X[i, ]`, ermöglicht das Ansprechen des i-ten Zeilenvektors, bzw. `X[, j]` für den j-ten Spaltenvektor.

```
> X[1,]
[1] 1 4
> X[,1]
[1] 1 2 3
> X[3,1]
[1] 3
> X[2:3,]
     [,1] [,2]
[1,]    2    5
[2,]    3    6
```

Das Gleiche gilt für dataframes.

Eigenschaften von Matrizen

Es ist auch möglich Zeilen und Spaltennamen zu vergeben:

```
> colnames(X) <- c("Apfel", "Birnen")
> rownames(X) <- c("Verkauf Montag","Verkauf Dienstag","Verkauf Mittwoch")
> X
                 Apfel Birnen
Verkauf Montag       1      4
Verkauf Dienstag     2      5
Verkauf Mittwoch     3      6
```

Die Namen beeinflussen so nicht den Datentyp:

```
> mode(X)
[1] "numeric"
```


Datenmatrix

Die Datenmatrix, in R `data.frame` genannt, wurde bereits in Teil I der Veranstaltung behandelt. Der wichtigste Unterschied zu Matrizen ist, dass pro Spalte unterschiedliche Strukturen vorliegen dürfen:

```
> str(miete)
'data.frame':        2053 obs. of  14 variables:
 $ nm          : num  741 716 528 554 698 ...
 $ nmqm        : num  10.9 11.01 8.38 8.52 6.98 ...
 $ wfl         : int  68 65 63 65 100 81 55 79 52 77 ...
 $ rooms       : int  2 2 3 3 4 4 2 3 1 3 ...
 $ bj          : num  1918 1995 1918 1983 1995 ...
 $ bez         : Factor w/ 25 levels "1","2","3","4",..: 2 2 2 16 16 16 6 6 6 6 ...
 $ wohngut     : Factor w/ 2 levels "0","1": 2 2 2 1 2 1 1 1 1 1 ...
 $ wohnbest    : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ ww0         : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ zh0         : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ badkach0    : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ badextra    : Factor w/ 2 levels "0","1": 1 1 1 2 2 1 2 1 1 1 ...
 $ kueche      : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1 1 ...
 $ roomsFaktor: Factor w/ 6 levels "ein","zwei","drei",..: 2 2 3 3 4 4 2 3 1 3 ...
```


Listen

Eine Liste erzeugt man mit dem Befehl `list(...)`, wobei `...` die Elemente der Liste enhält. Listen können beliebige Objekte enthalten, auch Objekte verschiedenen Typs:

```
> liste <- list(umsatz=100, Verkauf=X, Produkte=c("Apfel", "Birnen"))
> liste
$umsatz
[1] 100

$Verkauf
                 Apfel Birnen
Verkauf Montag       1      4
Verkauf Dienstag     2      5
Verkauf Mittwoch     3      6

$Produkte
[1] "Apfel"  "Birnen"
```


Datenanalyse Teil II

Einlesen von Daten

Als erstes immer das Arbeitsverzeichnis einstellen! Je nach vorliegendem Datenformat müssen dann verschiedene Funktionen benützt werden: Laden einer .RData-Datei:

```
> load("Daten/Kopie_miete.RData")
```

Text- und .csv-Dateien können mit den Funktionen `read.table()`, `read.csv()` und `read.csv2()` eingelesen werden, z.B.

```
> miete <- read.table("Daten/miete03.asc", header=TRUE)
```

Das Package `foreign` enthält Funktionen zum Einlesen von Formaten aus anderen Programmpaketen, wie z.B.`read.spss()`. Excel-Dateien lassen sich z.B. mit dem Package `xlsx` einlesen.

Einlesen von Daten

Es wird weiterhin der Miete-Datensatz verwendet:

```
> #setwd("Z:\\REinfuehrung")
> miete <- read.table("miete03.asc", header=TRUE)
> head(miete)
     nm  nmqm wfl rooms   bj bez wohngut wohnbest ww0 zh0 badkach0 badextra kueche
1 741.4 10.90  68     2 1918   2       1        0   0   0        0        0      0
2 715.8 11.01  65     2 1995   2       1        0   0   0        0        0      0
3 528.2  8.38  63     3 1918   2       1        0   0   0        0        0      0
4 554.0  8.52  65     3 1983  16       0        0   0   0        0        1      0
5 698.2  6.98 100     4 1995  16       1        0   0   0        0        1      1
6 935.6 11.55  81     4 1980  16       0        0   0   0        0        0      0
```


Datenstruktur

```
> str(miete)
'data.frame':        2053 obs. of  13 variables:
 $ nm      : num  741 716 528 554 698 ...
 $ nmqm    : num  10.9 11.01 8.38 8.52 6.98 ...
 $ wfl     : int  68 65 63 65 100 81 55 79 52 77 ...
 $ rooms   : int  2 2 3 3 4 4 2 3 1 3 ...
 $ bj      : num  1918 1995 1918 1983 1995 ...
 $ bez     : int  2 2 2 16 16 16 6 6 6 6 ...
 $ wohngut : int  1 1 1 0 1 0 0 0 0 0 ...
 $ wohnbest: int  0 0 0 0 0 0 0 0 0 0 ...
 $ ww0     : int  0 0 0 0 0 0 0 0 0 0 ...
 $ zh0     : int  0 0 0 0 0 0 0 0 0 0 ...
 $ badkach0: int  0 0 0 0 0 0 0 0 0 0 ...
 $ badextra: int  0 0 0 1 1 0 1 0 0 0 ...
 $ kueche  : int  0 0 0 0 1 0 0 0 0 0 ...
```

Sind kategoriale Variablen entsprechend kodiert?

Datenstruktur

Oft ist es sehr wichtig, dass kategoriale Variablen als Faktoren vorliegen. Deshalb werden die entsprechenden Variablen nun zu Faktoren umgewandelt. Die Stufen können übernommen oder vorgegeben werden.

```
> miete$badextra <- as.factor(miete$badextra)
> miete$badkach0 <- as.factor(miete$badkach0)
> miete$bez <- as.factor(miete$bez)
> miete$kueche <- as.factor(miete$kueche)
> miete$wohnbest <- as.factor(miete$wohnbest)
> miete$wohngut <- as.factor(miete$wohngut)
> miete$ww0 <- as.factor(miete$ww0)
> miete$zh0 <- as.factor(miete$zh0)
> miete$roomsFaktor <- factor(miete$rooms, levels=1:6,
+       labels=c("ein","zwei","drei","vier","fuenf","sechs"))
```

Datenstruktur

```
> summary(miete)
       nm              nmqm            wfl            rooms           bj
 Min.   :  77.3   Min.   : 1.47   Min.   : 17.0   Min.   :1.0   Min.   :1918
 1st Qu.: 389.9   1st Qu.: 6.80   1st Qu.: 53.0   1st Qu.:2.0   1st Qu.:1948
 Median : 534.3   Median : 8.47   Median : 67.0   Median :3.0   Median :1960
 Mean   : 570.1   Mean   : 8.39   Mean   : 69.6   Mean   :2.6   Mean   :1958
 3rd Qu.: 700.5   3rd Qu.:10.09   3rd Qu.: 83.0   3rd Qu.:3.0   3rd Qu.:1973
 Max.   :1789.5   Max.   :20.09   Max.   :185.0   Max.   :6.0   Max.   :2001

      bez       wohngut  wohnbest ww0      zh0      badkach0 badextra kueche
 9      : 177   0:1250   0:2008   0:1981   0:1878   0:1673   0:1862   0:1903
 2      : 161   1: 803   1:  45   1:  72   1: 175   1: 380   1: 191   1: 150
 5      : 139
 4      : 137
 3      : 132
 25     : 117
 (Other):1190
 roomsFaktor
 ein  :255
 zwei :715
 drei :759
 vier :263
 fuenf: 47
 sechs: 14
```


Datenstruktur

```
> str(miete)
'data.frame':        2053 obs. of  14 variables:
 $ nm          : num  741 716 528 554 698 ...
 $ nmqm        : num  10.9 11.01 8.38 8.52 6.98 ...
 $ wfl         : int  68 65 63 65 100 81 55 79 52 77 ...
 $ rooms       : int  2 2 3 3 4 4 2 3 1 3 ...
 $ bj          : num  1918 1995 1918 1983 1995 ...
 $ bez         : Factor w/ 25 levels "1","2","3","4",..: 2 2 2 16 16 16 6 6 6 6 ...
 $ wohngut     : Factor w/ 2 levels "0","1": 2 2 2 1 2 1 1 1 1 1 ...
 $ wohnbest    : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ ww0         : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ zh0         : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ badkach0    : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ badextra    : Factor w/ 2 levels "0","1": 1 1 1 2 2 1 2 1 1 1 ...
 $ kueche      : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1 1 ...
 $ roomsFaktor: Factor w/ 6 levels "ein","zwei","drei",..: 2 2 3 3 4 4 2 3 1 3 ...
```


Datenmanagement

Auswählen von Fällen oder Variablen:

- Auswahl der Wohnungen, die nach 1970 gebaut wurden und mindestens 70 m^2 Wohnfläche aufweisen:

```
> mieteJungGross <- miete[miete$bj > 1970 & miete$wfl >=70, ]
> dim(miete)
[1] 2053   14
> dim(mieteJungGross)
[1] 313  14
> mieteJungGross2 <- subset(miete, miete$bj > 1970 & miete$wfl >=70)
> dim(mieteJungGross2)
[1] 313  14
```

- Datensatz, der nur Angaben zu Nettomiete, Wohnfläche und Zimmeranzahl enthält:

```
> mieteNmWflRooms <- subset(miete, select=c(nm, wfl, rooms))
> dim(mieteNmWflRooms)
[1] 2053    3
```


Datenmanagement

- Datensatz, der keine Angaben zum Bezirk enthält:

```
> mieteOhneBez <- subset(miete, select=-bez)
> dim(mieteOhneBez)
[1] 2053   13
```

Reproduzierbarkeit: Immer Rohdaten zusammen mit Skript aller notwendigen Transformationen (aus Skriptfenster) speichern! Für weitere Bearbeitung ist zusätzliches Speichern der fertigen Datendatei als .RData-Datei praktisch.

Variablen hinzufügen

Erzeugen einer neuen Variable aus bestehenden (oder Umbenennung): Es stehen alle R-Funktionen zur Verfügung, häufig werden jedoch die Grundrechnungsarten und einfache Funktionen wie `log()` oder `exp()` verwendet. Es können auch mehrere Variablen verknüpft werden.

```
> miete$wflProRaum <- miete$wfl / miete$rooms
> summary(miete$wflProRaum)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   12.0    23.5    26.7    27.8    31.0   146.0
```


Lage, Streuung, Quantile

Kennzahlen des numerischen Merkmals Nettomiete:

```
> summary(miete$nm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   77.3   390.0   534.0   570.0   700.0  1790.0
> sd(miete$nm)
[1] 245.4
```

Kennzahlen sind auch einzeln verfügbar:

```
> min(miete$nm)
[1] 77.31
> max(miete$nm)
[1] 1790
> mean(miete$nm)
[1] 570.1
> median(miete$nm)
[1] 534.3
> quantile(miete$nm, prob=0.75)
  75%
700.5
```


Fehlende Werte

Liegen fehlende Werte vor, kann dies zu Problemen führen. In R sind fehlende Werte mit `NA` (Not Available) codiert. `NA` kann sich aus der Datenerhebung ergeben, kann das Ergebnis einer (fehlgeschlagenen) Berechnung sein oder auch zugewiesen werden. Mit der Funktion `is.na()` kann überprüft werden, wo fehlende Werte auftreten. Bei einigen Funktionen können diese mittels des Arguments `na.rm` ausgeschlossen werden. Weitere Behandlungsmöglichkeiten für fehlende Werte stellen u.a. die Funktionen `na.omit()` und `complete.cases()` zur Verfügung.

```
> fehl <- c(1, 2, NA, 3)
> is.na(fehl)
[1] FALSE FALSE  TRUE FALSE
> mean(fehl)
[1] NA
> mean(fehl, na.rm=TRUE)
[1] 2
```


Bivariate Deskription

Balkendiagramme für Kontingenztafeln

Anzahl der Zimmer je nachdem, ob die Wohnfläche mehr als 80 m^2 beträgt:

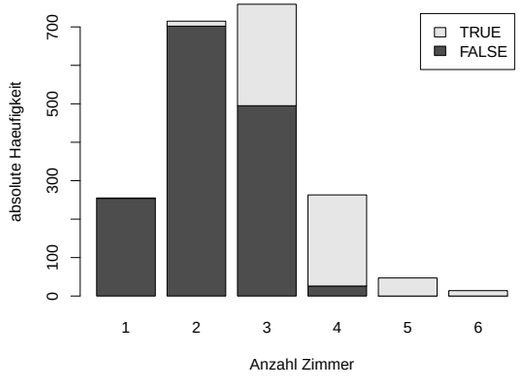
```
> TAB <- table(miete$wfl>80, miete$rooms)
> TAB
         1   2   3   4   5   6
  FALSE 254 702 495  26   0   0
  TRUE    1  13 264 237  47  14
```

Die Anteile je Zeile bzw. Spalte erhhält man durch:

```
> prop.table(TAB, margin=1)
              1        2        3        4        5        6
  FALSE 0.171970 0.475288 0.335139 0.017603 0.000000 0.000000
  TRUE  0.001736 0.022569 0.458333 0.411458 0.081597 0.024306
> TAB2 <- prop.table(TAB, margin=2)
> TAB2
              1        2        3        4        5        6
  FALSE 0.996078 0.981818 0.652174 0.098859 0.000000 0.000000
  TRUE  0.003922 0.018182 0.347826 0.901141 1.000000 1.000000
```

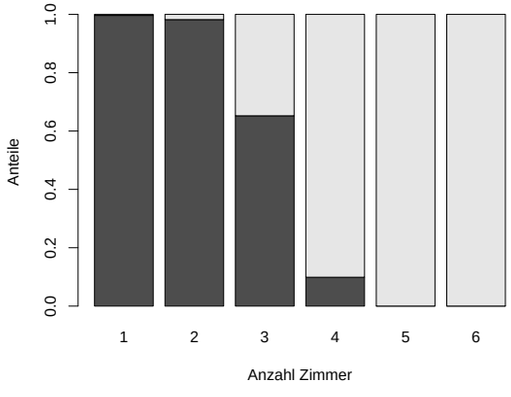

Balkendiagramme für Kontingenztafeln

```
> barplot(TAB, legend=TRUE, xlab="Anzahl Zimmer", ylab="absolute Haeufigkeit")
```



Balkendiagramme für Kontingenztafeln

```
> barplot(TAB2, xlab="Anzahl Zimmer", ylab="Anteile")
```



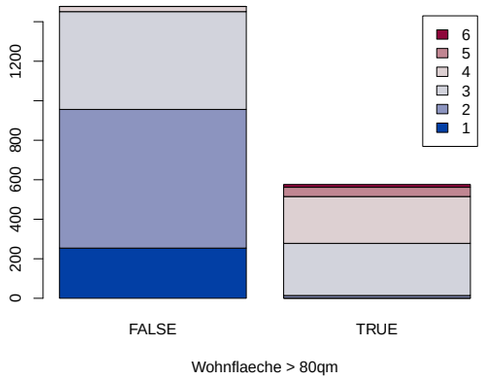
Balkendiagramme für Kontingenztafeln

Die Funktion `t()` transponiert eine Matrix:

```
> TAB
         1   2   3   4   5   6
  FALSE 254 702 495  26   0   0
  TRUE    1  13 264 237  47  14
> t(TAB)
    FALSE TRUE
1     254    1
2     702   13
3     495  264
4      26  237
5       0   47
6       0   14
```

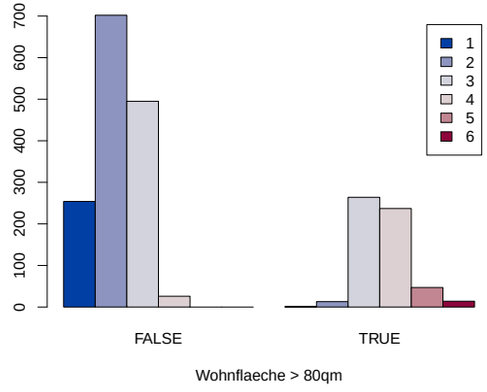
Balkendiagramme für Kontingenztafeln

```
> library(colorspace)
> barplot(t(TAB), legend=TRUE, col=diverge_hcl(6), xlab="Wohnflaeche > 80qm")
```



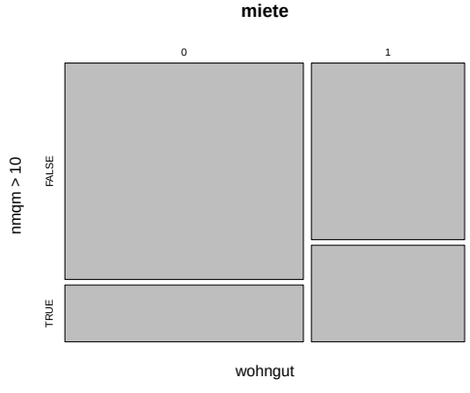
Balkendiagramme für Kontingenztafeln

```
> barplot(t(TAB), beside=TRUE, legend=TRUE, col=diverge_hcl(6), xlab="Wohnflaeche > 80qm")
```



Mosaicplot: Miete, Lage

```
> mosaicplot(~wohngut + (nmqm > 10), data = miete)
```



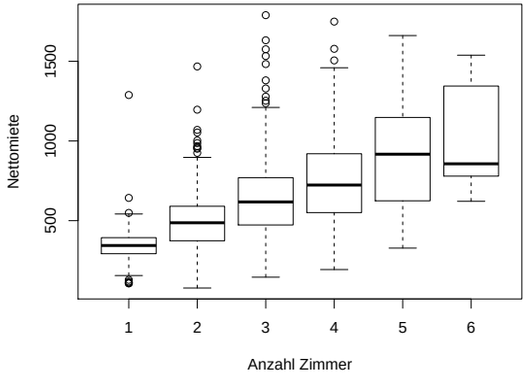
Mosaicplot: Miete, Lage

Welche Zahlen stehen hinter den Flächen des Mosaicplots?

```
> table(miete$nmqm>10, miete$wohngut)
         0   1
  FALSE 990 519
  TRUE  260 284
```

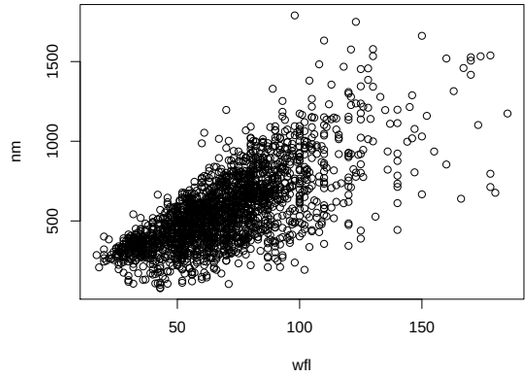
Boxplot: Nettomiete nach Anzahl Zimmer

```
> boxplot(nm ~ rooms, xlab="Anzahl Zimmer", ylab="Nettomiete", data=miete)
```



Streudiagramm: Nettomiete und Fläche

```
> plot(nm ~ wfl, data=miete)
> # plot(x=miete$wfl, y=miete$nm) # Alternative
```



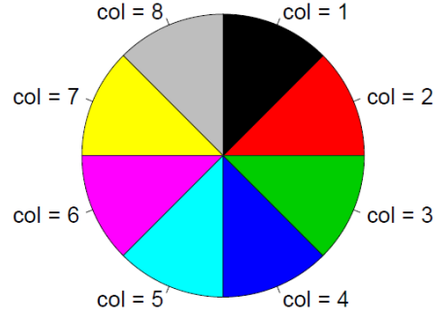
Argumente für Grafikfunktionen

Eine ausführliche Hilfe zu den Graphikparametern findet man mit `?par()`. Hier eine Zusammenfassung von wichtigen Argumenten (Auswahl):

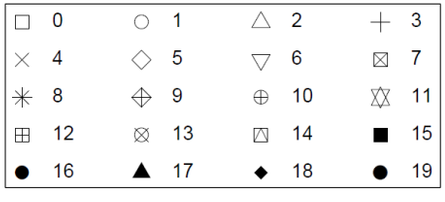
Argument	Beschreibung
`main`	Überschrift der Grafik
`xlab, ylab`	Name(„Label“) der x-/y-Achsen
`xlim, ylim`	Vektor mit Minimum/Maximum für zu plottenden Bereich in x-/y-Richtung
`cex`	Größe
`cex.main, cex.axis, cex.lab`	Größe der Überschrift, Achsenbeschriftung und -namen
`col`	Farbe der Objekte in der Plot-Region Zahlen: 1(Schwarz), 2(rot),... Übersicht mit `palette()` Namen: “black“, “red“,... Übersicht mit `colours()`
`lty`	Linientyp (z.B. gestrichelt,...)
`lwd`	Linienbreite
`type`	“p“: Punkte, “l“: Linien, “n“: Nichts einzeichen
`pch`	Symbol für Punkte (1: Kreis, 2: Dreieck,...)

Argumente für Grafikfunktionen

col:



pch:



Elemente hinzufügen

Manchmal möchte man Elemente zu einer bestehenden Graphik hinzufügen. Z. B. könnte man den Mittelwert mit `abline()` in ein Histogramm einzeichnen.

Funktion	Beschreibung
`points()`	Punkte an Stellen (x, y)
`lines()`	Linien zwischen den Stellen (x, y)
`segments()`	Liniensegmente
`arrows()`	ähnlich wie `segements`, aber mit Pfeilspitzen
`text()`	Text
`title()`	Beschriftung
`axis()`	Achsen hinzufügen; x-Achse: `side="1"`, y-Achse: `side="2"`
`abline()`	Eine oder mehrere Geraden
`grid()`	Gitternetz

Schichtung

Kennzahlen eines numerischen Merkmals geschichtet nach einem kategorialen Merkmal:

```
> aggregate(x=miete$nm, by=list(miete$roomsFaktor), FUN=summary)
  Group.1 x.Min. x.1st Qu. x.Median x.Mean x.3rd Qu. x.Max.
1     ein  106.0     293.0    344.0  347.0     393.0 1290.0
2    zwei   77.3     374.0    487.0  487.0     590.0 1470.0
3    drei  145.0     473.0    618.0  634.0     770.0 1790.0
4    vier  193.0     551.0    723.0  748.0     920.0 1750.0
5   fuenf  328.0     625.0    917.0  901.0    1150.0 1660.0
6   sechs  622.0     781.0    857.0  999.0    1300.0 1540.0
```


Korrelation

Korrelation nach Pearson:

```
> cor(miete[,c("nm","nmqm","wfl")], use="complete.obs", method="pearson")
         nm    nmqm     wfl
nm   1.0000  0.4748  0.7075
nmqm 0.4748  1.0000 -0.2268
wfl  0.7075 -0.2268  1.0000
```

Korrelation nach Spearman:

```
> cor(miete[,c("nm","nmqm","wfl")], use="complete.obs", method="spearman")
         nm    nmqm     wfl
nm   1.0000  0.4611  0.6971
nmqm 0.4611  1.0000 -0.2304
wfl  0.6971 -0.2304  1.0000
```

Ein Test auf die Korrelation zwischen zwei Variablen kann mit `cor.test` durchgeführt werden.

Lineare Regression

```
> RegModel1 <- lm(nm ~ wfl, data=miete)
> summary(RegModel1)
Call:
lm(formula = nm ~ wfl, data = miete)

Residuals:
   Min     1Q Median     3Q    Max
-655.2  -97.4    7.4   98.7 1023.4

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   89.847     11.264    7.98  2.5e-15 ***
wfl            6.901      0.152   45.33  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 174 on 2051 degrees of freedom
Multiple R-squared:  0.501,        Adjusted R-squared:   0.5
F-statistic: 2.06e+03 on 1 and 2051 DF,  p-value: <2e-16
```

Dokumente erstellen

Grafiken

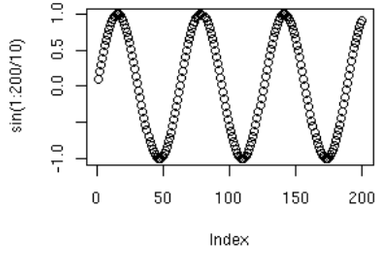
Es gibt zwei grundsätzlich verschiedene Arten, Grafiken zu speichern:

Bitmap: In einem rechteckigen Gitter wird für jeden Gitterpunkt gespeichert, welche Farbe er hat. Reskalierungen reduzieren Qualität.
Bekannte Formate: JPEG, BMP, PNG, GIF

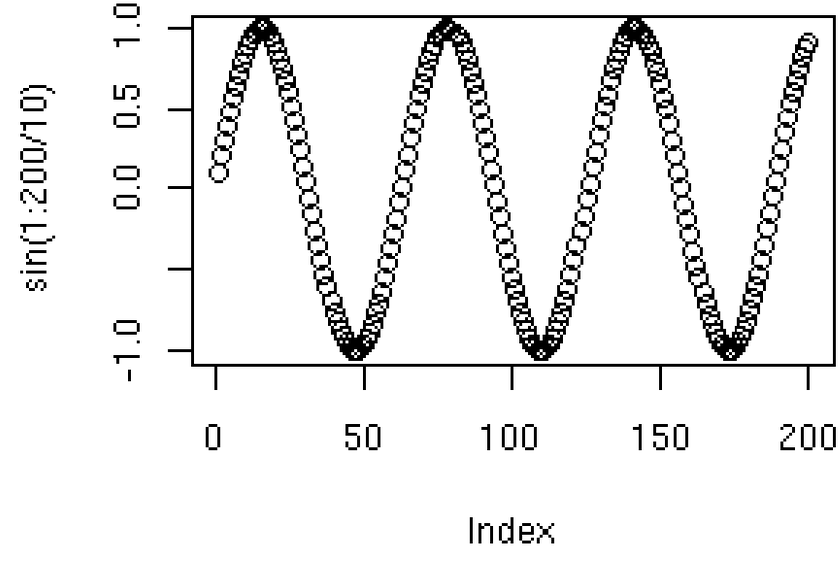
Vektor-Grafik: Eine logische Beschreibung der Grafik, ohne Qualitätsverlust reskalierbar.
Bekannte Formate: EMF, WMF, EPS, PDF

Der große Vorteil der Vektor-Grafiken gegenüber Bitmap-Grafiken liegt in der Skalierbarkeit ohne Qualitätsverslust (d. h. die Bilder werden nicht pixelig). Bitmap-Grafiken lassen sich dafür oft einfacher in andere Software einbinden und betrachten. Ein PDF-Bild kann man in Word nur als „Objekt“, aber nicht als „Grafik“ einbetten.

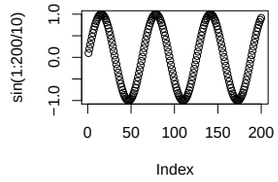
Bitmap vs. Vektorgrafik



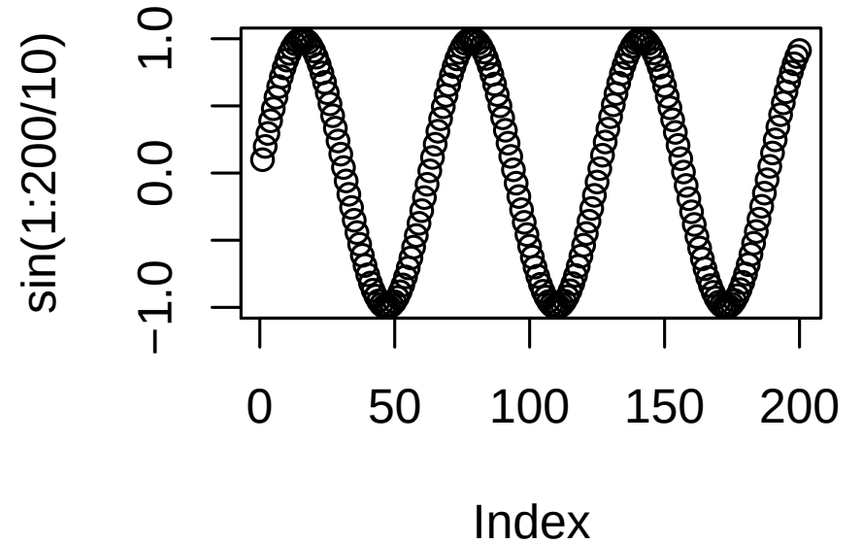
Bitmap vs. Vektorgrafik



Bitmap vs. Vektorgrafik



Bitmap vs. Vektorgrafik



Textverarbeitung

Wenn Ausgaben aus der R-Konsole in ein Textverarbeitungsprogramm (z. B. Word) kopiert werden, ist es meist sinnvoll, diese in einer Schriftart mit fixen Zeichenabständen (Nicht-Proportionalschrift) zu formatieren (z.B. Courier, Typewriter, . . .):

Times New Roman (Proportionalschrift):

```
TAB <- table(miete$wfl>80, miete$rooms)
> TAB

      1  2  3  4  5  6
 FALSE 254 702 495 26  0  0
 TRUE   1 13 264 237 47 14
```

Courier New (Nicht-Proportionalschrift):

```
TAB <- table(miete$wfl>80, miete$rooms)
> TAB

          1   2   3   4   5   6
  FALSE 254 702 495  26   0   0
  TRUE    1  13 264 237  47  14
```


Hausübung

In der Hausübung arbeitet jede/r mit einem eigenen Datensatz. Dieser muss durch eine Zufallsstichprobe aus einem Datensatz (data_all) erzeugt werden. Dazu verwendet jede/r die eigene (!) Matrikelnummer als Seed. Der übrige R-Code kann so übernommen werden:

```
> matrikelnummer <-  11111111
> set.seed(matrikelnummer)
> lines_sample <- sample(x=1:nrow(data_all), size=200, replace = FALSE, prob = NULL)
> data <- data_all[lines_sample,]
```

Mit `sample()` werden 200 verschiedene Zahlen (zwischen 1 und Anzahl der Beobachtungen) zufällig gezogen. Anschließend werden die den Zahlen entsprechenden Beobachtungszeilen im Datensatz ausgewählt. Der so erstellte individuelle (Teil-)Datensatz muss bei der Bearbeitung der Hausübung bei allen drei Software-Paketen verwendet werden. Die Anforderungen an die Daten können sich je nach Software/Plattform etc. unterscheiden (z. B. Dezimaltrennzeichen). Deshalb immer den gespeicherten Datensatz verwenden, der richtig funktioniert.