

Statistik-Grundlagen mit R

Datenaufbereitung

Fehlende Werte (NA)

<code>is.na(x)</code>	TRUE für jeden fehlenden Wert
<code>anyNA(x)</code>	Gibt es mindestens einen fehlenden Wert?
<code>sum(is.na(x))</code>	Anzahl der fehlenden Werte
<code>is.finite(x)</code>	FALSE für NA, NaN und Inf
<code>complete.cases(df)</code>	TRUE für jede Zeile ohne fehlende Werte
<code>na.omit(df)</code>	Entfernt Zeilen mit fehlenden Werten
<code>na.rm = TRUE</code>	Argument der meisten Zusammenfassungsfunktionen, um NAs zu ignorieren, z. B. <code>mean(x, na.rm = TRUE)</code>
<code>cor(x, y, use = "complete.obs")</code>	Verwendet in <code>cor()</code> und <code>cov()</code> nur vollständige Beobachtungen

Modellfunktionen wie `lm()` entfernen unvollständige Zeilen standardmäßig (`na.action = na.omit`).

Zufallszahlen und Stichproben

<code>set.seed(n)</code>	Setzt den Startwert des Zufallszahlengenerators für reproduzierbare Ergebnisse
<code>sample(x, size)</code>	Zufallsstichprobe ohne Zurücklegen; <code>sample(x)</code> liefert eine zufällige Permutation
<code>sample(x, size, replace = TRUE)</code>	Zufallsstichprobe mit Zurücklegen (z. B. Bootstrap)
<code>replicate(n, expr)</code>	Wiederholt einen Ausdruck <code>n</code> -mal, z. B. für Simulationen
<code>rnorm(n, mean, sd)</code>	Zufallszahlen aus einer Verteilung (alle Verteilungen: siehe <i>Wahrscheinlichkeitsverteilungen</i>)

Transformationen

Mathematische Funktionen

<code>round(x, n)</code>	Rundet die Elemente von <code>x</code> auf <code>n</code> Dezimalstellen (.5 wird zur geraden Ziffer gerundet)
<code>signif(x, n)</code>	Rundet die Elemente von <code>x</code> auf <code>n</code> signifikante Stellen
<code>ceiling(x)</code>	Rundet auf die kleinste ganze Zahl, die nicht kleiner als das jeweilige Element von <code>x</code> ist
<code>floor(x)</code>	Rundet auf die größte ganze Zahl, die nicht größer als das jeweilige Element von <code>x</code> ist
<code>trunc(x)</code>	Rundet durch Abschneiden der Nachkommastellen (in Richtung 0)
<code>abs(x)</code>	Berechnet die Absolutwerte von <code>x</code>
<code>sqrt(x)</code>	Berechnet die (Haupt-)Quadratwurzel von <code>x</code>
<code>x^(1/3)</code>	Berechnet die Kubikwurzel von <code>x</code>
<code>log(x, base)</code>	Berechnet Logarithmen zur Basis <code>base</code> (Standard ist der natürliche Logarithmus)
<code>log2(x)</code>	Berechnet den binären Logarithmus (Basis 2)
<code>log10(x)</code>	Berechnet den Logarithmus zur Basis 10
<code>log1p(x)</code>	Berechnet $\log(1+x)$
<code>exp(x)</code>	Berechnet die Exponentialfunktion
<code>expm1(x)</code>	Berechnet $\exp(x) - 1$
<code>car::basicPower()</code>	Berechnet eine einfache Potenztransformation
<code>car::bcPower()</code>	Transformiert die Elemente von <code>x</code> mit der Box-Cox-Familie skalierten Potenztransformationen
<code>sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), atan2(x)</code>	Trigonometrische Funktionen; sie berechnen Sinus, Kosinus, Tangens, Arkussinus, Arkuskosinus, Arkustangens und den Arkustangens mit zwei Argumenten.
<code>scale(x)</code>	Zentriert und/oder skaliert einen Vektor oder eine Matrix <code>x</code> ; nur zentrieren mit <code>scale=FALSE</code> , nur skalieren mit <code>center=FALSE</code> ; Standard ist <code>center=TRUE</code> , <code>scale=TRUE</code> für die <i>z-Transformation</i>

Transformation auswählen

Daten	Transformation
Rechtsschiefe, positive Werte	<code>log(x)</code> ; <code>log1p(x)</code> , falls Nullen vorkommen
Zählraten	<code>sqrt(x)</code>
Anteile (0-1)	<code>asin(sqrt(p))</code> (Arkussinus) oder <code>qlogis(p)</code> (Logit)
Unbekannte Potenz	<code>MASS::boxcox(model)</code> , um λ für <code>car::bcPower()</code> zu finden
Unterschiedliche Einheiten oder Skalen	<code>scale(x)</code> (z-Transformation)
Werte auf den Bereich 0-1	<code>(x-min(x))/diff(range(x))</code>

Rücktransformation: `exp()` bei `log`, `x^2` bei `sqrt`, `sin(y)^2` bei Arkussinus, `plogis()` bei Logit. Nach der Transformation die Modellannahmen erneut prüfen.

Matrixalgebra

Matrixfunktionen

<code>t(x)</code>	Transponiert <code>x</code>
<code>diag(x)</code>	Erzeugt eine Diagonalmatrix (bzw. extrahiert die Diagonale einer Matrix)
<code>a %*% b</code>	Matrixmultiplikation
<code>crossprod(x)</code>	Kreuzprodukt <code>t(x) %*% x</code> (schneller)
<code>det(a)</code>	Determinante von <code>a</code>
<code>solve(a)</code>	Gibt die inverse Matrix von <code>a</code> zurück
<code>solve(a,b)</code>	Löst $a \%*\% x = b$ nach <code>x</code>
<code>eigen(a)</code>	Eigenwerte und Eigenvektoren von <code>a</code>
<code>outer(x, y)</code>	Äußeres Produkt zweier Vektoren

Beispiel: Kleinste-Quadrate-Schätzer eines linearen Modells mit Designmatrix `X` und Zielvariable `y`:

```
solve(crossprod(X), crossprod(X, y))
```

Deskriptive Statistik

Funktionen mit einem einzelnen Wert als Ergebnis

<code>sum(x)</code>	Summe der Elemente von x
<code>prod(x)</code>	Produkt der Elemente von x
<code>mean(x)</code>	Arithmetisches Mittel der Elemente von x; mit <code>trim = 0.1</code> ein getrimmtes Mittel
<code>weighted.mean(x, w)</code>	Mittelwert von x mit Gewichten w
<code>median(x)</code>	Median der Elemente von x
<code>var(x)</code>	Varianz der Elemente von x (berechnet mit $n - 1$)
<code>sd(x)</code>	Standardabweichung von x
<code>sd(x)/sqrt(length(x))</code>	Berechnung des Standardfehlers des Mittelwerts
<code>sd(x)/mean(x)</code>	Variationskoeffizient (relative Streuung)
<code>min(x)</code>	Minimum der Elemente von x
<code>max(x)</code>	Maximum der Elemente von x
<code>length(x)</code>	Anzahl der Elemente von x
<code>sum(!is.na(x))</code>	Anzahl der nicht fehlenden Werte
<code>names(which.max(table(x)))</code>	Modus (häufigster Wert)

Mit `na.rm = TRUE` werden fehlende Werte ignoriert.

Funktionen mit Einzelwerten o. Matrizen als Ergebnis

<code>summary(x)</code>	Für einen numerischen Vektor: Minimum, Quartile, Median, Mittelwert und Maximum (plus Anzahl der NAs); bei einer Matrix oder einem Data Frame werden diese Kennzahlen für jede Spalte berechnet
<code>cor(x, y)</code>	Korrelation zwischen x und y bzw. Korrelationsmatrix, falls es sich um Matrizen oder Data Frames handelt; das Argument <i>method</i> wählt zwischen Pearsons Produkt-Moment-Korrelationskoeffizient (Standard), Kendalls Tau und Spearmans Rho
<code>cov(x, y)</code>	Kovarianz zwischen x und y

Funktionen mit einem Vektor als Ergebnis

<code>range(x)</code>	Gibt Minimum und Maximum der Elemente von x zurück
<code>pmin(x,y), pmax(x,y)</code>	Gibt parallele (elementweise) Minima/Maxima über mehrere Vektoren zurück
<code>quantile(x, probs)</code>	Stichprobenquantile zu den angegebenen Wahrscheinlichkeiten, z. B. <code>probs = c(0.25, 0.75)</code>
<code>fivenum(x)</code>	Fünf-Zahlen-Zusammenfassung nach Tukey: Minimum, unteres Hinge, Median, oberes Hinge, Maximum
<code>diff(x)</code>	Verzögerte und iterierte Differenzen des Vektors x
<code>rank(x)</code>	Ränge der Elemente von x
<code>IQR(x)</code>	Interquartilsabstand der Werte von x
<code>mad(x)</code>	Median der absoluten Abweichungen
<code>cumsum(x)</code>	Kumulierte Summen der Elemente von x
<code>cumprod(x)</code>	Kumuliertes Produkt der Elemente von x
<code>cummin(x)</code>	Kumulierte Minima der Elemente von x
<code>cummax(x)</code>	Kumulierte Maxima der Elemente von x
<code>colSums(x)</code>	Summe jeder Spalte
<code>rowSums(x)</code>	Summe jeder Zeile
<code>colMeans(x)</code>	Arithmetisches Mittel jeder Spalte
<code>rowMeans(x)</code>	Arithmetisches Mittel jeder Zeile

Häufigkeiten und Tabellen

<code>table(x)</code>	Häufigkeitstabelle: Anzahl jedes Werts bzw. Levels von x
<code>table(x, y)</code>	Kreuztabelle (Kontingenztable) zweier Variablen
<code>prop.table(t)</code>	Relative Häufigkeiten; <code>margin = 1</code> liefert Anteile je Zeile, <code>margin = 2</code> je Spalte
<code>addmargins(t)</code>	Fügt einer Tabelle Zeilen- und Spaltensummen hinzu
<code>xtabs(~ a + b, data)</code>	Kreuztabelle aus einer Formel und einem Data Frame
<code>cumsum(table(x))</code>	Kumulierte Häufigkeiten
<code>cut(x, breaks)</code>	Teilt eine numerische Variable in Klassen ein (Faktor), z. B. vor <code>table()</code>

Gruppenweise Kennzahlen

<code>tapply(x, g, mean)</code>	Wendet eine Funktion (hier den Mittelwert) getrennt für jede Gruppe in g auf x an
<code>aggregate(y ~ g, data, mean)</code>	Dasselbe mit einer Formel; gibt einen Data Frame zurück (mehrere Gruppen: <code>y ~ g1 + g2</code> , alle anderen Spalten: <code>. ~ g</code>)
<code>split(x, g)</code>	Teilt x in eine Liste mit einem Element je Gruppe auf, z. B. <code>sapply(split(x, g), sd)</code>
<code>table(g)</code>	Anzahl der Beobachtungen je Gruppe

Form einer Verteilung und Ausreißer

Rechtsschief	Langer Auslauf nach rechts; Mittelwert > Median
Linksschief	Langer Auslauf nach links; Mittelwert < Median
Symmetrisch	Mittelwert $\approx$ Median
<code>e1071::skewness(x)</code>	Schiefekoeffizient (> 0: rechtsschief, < 0: linksschief)
<code>e1071::kurtosis(x)</code>	Kurtosis (Wölbung und Gewicht der Verteilungsenden)

Faustregel für Ausreißer (Boxplot): Werte unter $Q1 - 1,5 \cdot IQR$ oder über $Q3 + 1,5 \cdot IQR$.

Grafische Zusammenfassungen in Base R

<code>hist(x)</code>	Histogramm (Klassenanzahl über <i>breaks</i>)
<code>plot(density(x))</code>	Kerndichteschätzung (geglättetes Histogramm)
<code>boxplot(y ~ g, data)</code>	Boxplot je Gruppe
<code>stripchart(y ~ g, data)</code>	Darstellung der einzelnen Werte je Gruppe
<code>barplot(table(x))</code>	Balkendiagramm der Häufigkeiten
<code>plot(x, y)</code>	Streudiagramm zweier Variablen
<code>pairs(df)</code>	Streudiagramm-Matrix aller Spaltenpaare
<code>par(mfrow=c(1,2))</code>	Ordnet mehrere Plots (Zeilen, Spalten) in einem Fenster an
<code>boxplot.stats(x)\$out</code>	Werte außerhalb der Whisker (mögliche Ausreißer)

Inferenzstatistik

Grundbegriffe des Hypothesentestens:

- **H₀**: Nullhypothese (kein Effekt, kein Unterschied); **H₁**: Alternativhypothese
- **p-Wert**: Wahrscheinlichkeit des beobachteten (oder eines extremeren) Ergebnisses, wenn H₀ gilt; H₀ wird verworfen, wenn $p < \alpha$ (meist 0,05)
- **Fehler 1. Art** (α): Ablehnen einer wahren H₀; **Fehler 2. Art** (β): Beibehalten einer falschen H₀; **Power** = $1 - \beta$
- **Standardfehler**: Standardabweichung einer Schätzung, z. B. des Mittelwerts: $sd/\sqrt{n}$
- **Konfidenzintervall**: Bereich plausibler Werte für einen Parameter (95% solcher Intervalle enthalten bei wiederholter Stichprobenziehung den wahren Wert)
- Das Argument `alternative` legt einen zweiseitigen ("two.sided") oder einseitigen ("less", "greater") Test fest

Wahrscheinlichkeitsverteilungen

Gauß-(Normal-)Verteilung:

<code>rnorm()</code>	Erzeugt Zufallszahlen (rxxx)
<code>pnorm(q)</code>	Gibt die Wahrscheinlichkeiten für gegebene Quantile zurück (pxxx)
<code>qnorm(p)</code>	Gibt die Quantile für gegebene Wahrscheinlichkeiten zurück (qxxx)
<code>dnorm(x)</code>	Liefert die Dichtefunktion (dxxx)

Weitere Standardverteilungen:

(siehe auch `?Distributions`; `d` durch `r`, `p` oder `q` ersetzen, wie bei der Normalverteilung)

<code>dpois(x, lambda)</code>	Poisson
<code>dnbinom(x, size, prob)</code>	Negative Binomial
<code>dbinom(x, size, prob)</code>	Binomial
<code>dmultinom(x, size, prob)</code>	Multinomial
<code>dunif(x, min=0, max=1)</code>	Gleichverteilung
<code>dexp(x, rate=1)</code>	Exponential
<code>dlnorm(x, meanlog=0, sdlog=1)</code>	Lognormal
<code>dgamma(x, shape, scale=1)</code>	Gamma
<code>dbeta(x, shape1, shape2)</code>	Beta
<code>dt(x, df)</code>	Student-t-Verteilung
<code>df(x, df1, df2)</code>	F-Verteilung
<code>dchisq(x, df)</code>	(Nichtzentrale) Chi-Quadrat-Verteilung
<code>dweibull(x, shape, scale=1)</code>	Weibull
<code>dcauchy(x, location=0, scale=1)</code>	Cauchy
<code>dgeom(x, prob)</code>	Geometrisch

Tests für kategoriale Daten und Anteile

<code>chisq.test(table)</code>	Chi-Quadrat-Unabhängigkeitstest für eine Kontingenztafel (erwartete Häufigkeiten sollten ≥ 5 sein)
<code>chisq.test(x, p=p)</code>	Chi-Quadrat-Anpassungstest gegen vorgegebene Wahrscheinlichkeiten
<code>fisher.test(table)</code>	Exakter Test nach Fisher (kleine Häufigkeiten)
<code>prop.test(x, n)</code>	Test und Konfidenzintervall für einen oder mehrere Anteile
<code>binom.test(x, n, p)</code>	Exakter Binomialtest für einen Anteil

Tests der Modellannahmen für numerische Daten

<code>shapiro.test(x)</code>	Führt den Shapiro-Wilk-Test auf Normalverteilung durch
<code>ks.test(x,y)</code>	Führt einen Ein- oder Zwei-Stichproben-Kolmogorov-Smirnov-Test durch
<code>var.test(x,y)</code>	Führt einen F-Test zum Vergleich der Varianzen zweier Stichproben aus normalverteilten Grundgesamtheiten durch
<code>bartlett.test()</code>	Führt den Bartlett-Test auf Varianzhomogenität zwischen Gruppen durch
<code>fligner.test()</code>	Führt den Fligner-Killeen-Test auf Varianzhomogenität durch (robust gegenüber Abweichungen von der Normalverteilung)
<code>car::leveneTest(y, group)</code>	Berechnet den Levene-Test auf Varianzhomogenität zwischen Gruppen
<code>car::durbinWatsonTest()</code>	Führt den Durbin-Watson-Test auf autokorrelierte Fehler durch

Tests hängen von der Stichprobengröße ab: Bei großen Stichproben werden schon kleine Abweichungen signifikant, bei kleinen werden sie übersehen. Immer auch grafisch prüfen (`hist()`, `qqnorm()`, `boxplot()`).

Einfache Tests für numerische Daten

Ein- und Zwei-Stichproben-Tests

<code>t.test(x)</code>	Einstichproben-t-Test; <code>mu</code> legt den Vergleichswert fest, <code>conf.level</code> das Konfidenzniveau
<code>t.test(x,y)</code>	Zweistichproben-t-Test für unabhängige Stichproben (standardmäßig Welch-Test; für den t-Test nach Student <code>var.equal=TRUE</code> setzen)
<code>t.test(x,y, paired=TRUE)</code>	Zweistichproben-t-Test für abhängige Stichproben
<code>t.test(y ~ g, data)</code>	Zweistichproben-t-Test mit Formel und Gruppierungsvariable
<code>t.test(x)\$conf.int</code>	95%-Konfidenzintervall des Mittelwerts
<code>wilcox.test(x)</code>	Nichtparametrischer Wilcoxon-Vorzeichen-Rang-Test (eine Stichprobe)
<code>wilcox.test(x,y)</code>	Nichtparametrischer Mann-Whitney-U-Test für zwei Stichproben (bzw. Wilcoxon-Rangsummentest)
<code>wilcox.test(x,y, paired=TRUE)</code>	Nichtparametrischer Wilcoxon-Vorzeichen-Rang-Test für zwei abhängige Stichproben

Tests für mehrere Stichproben

<code>kruskal.test(x, g)</code>	Führt den nichtparametrischen Kruskal-Wallis-Rangsummentest durch
<code>friedman.test(y, groups)</code>	Führt den Friedman-Rangsummentest für unreplicierte Blockdaten durch

Korrelationstests

<code>cor.test(x, y)</code>	Test auf Korrelation zwischen gepaarten Stichproben; mit <code>method = "spearman"</code> oder <code>"kendall"</code> für Rangkorrelationen
-----------------------------	---

Multiples Testen und Post-hoc-Tests

<code>p.adjust(p, method="holm")</code>	Korrigiert p-Werte für multiple Vergleiche; weitere Methoden: "bonferroni", "BH" (False Discovery Rate)
<code>pairwise.t.test(x, g)</code>	Paarweise t-Tests zwischen allen Gruppen (p-Werte korrigiert)
<code>pairwise.wilcox.test(x, g)</code>	Paarweise Wilcoxon-Tests zwischen allen Gruppen

Power-Analyse

Berechnet die Power oder bestimmt Parameter (z. B. den Stichprobenumfang), um eine Ziel-Power zu erreichen; der zu berechnende Parameter wird als NULL angegeben:

Base R (Paket *stats*)

<code>power.t.test()</code>	Ein- oder Zwei-Stichproben-t-Test
<code>power.anova.test()</code>	Balancierte einfaktorielle ANOVA
<code>power.prop.test()</code>	Zweistichproben-Test für Anteile

Weitere Pakete

<code>pwr::pwr.t.test()</code>	t-Tests (1-/2-Stichproben, unabhängig, gepaart)
<code>pwr::pwr.t2n.test()</code>	t-Test (2 Stichproben, ungleiches n)
<code>MKpower::power.welch.t.test()</code>	Welch-t-Test
<code>WMWssp::WMWssp()</code>	Mann-Whitney-U-Test (nur Berechnung von n ; benötigt Daten einer Referenzgruppe)
<code>pwr::pwr.chisq.test()</code>	Chi-Quadrat-Test
<code>pwr::pwr.anova.test()</code>	Balancierte einfaktorielle ANOVA (gleiches n)
<code>MultNonParam::kwpower()</code>	Kruskal-Wallis-Test
<code>MultNonParam::kwsamplesize()</code>	
<code>pwr::pwr.r.test()</code>	Korrelation
<code>WebPower::wp.regression()</code>	Lineare Regression

Lineare Modelle

Ein Rahmen: das lineare Modell

Erklärende Variablen	Bezeichnung und Formel
eine kontinuierliche	Einfache lineare Regression: $y \sim x$
mehrere kontinuierliche	Multiple Regression: $y \sim x_1 + x_2$
eine oder mehrere kategoriale (Faktoren)	ANOVA (ein- oder mehrfaktoriell): $y \sim f_1 + f_2$
kontinuierliche und kategoriale	ANCOVA: $y \sim x + f$ (parallele Geraden), $y \sim x * f$ (unterschiedliche Steigungen)

Alle diese Modelle sind lineare Modelle. `lm()` und `aov()` passen das selbe Modell an und unterscheiden sich vor allem in der Standardausgabe: `summary(model)` zeigt die Koeffizienten (bei Faktoren: Unterschiede zum Referenzlevel), `anova(model)` die Varianztabelle je Term.

Anpassen und Ergebnisse

<code>lm(y ~ x, data)</code>	Passt ein lineares Modell an; die Zielvariable Y muss kontinuierlich sein, die erklärenden Variablen X können kontinuierlich oder kategorial sein (die Funktion kann für lineare Regressionen, ANCOVAs und ANOVAs verwendet werden)
<code>aov(y ~ x, data)</code>	Passt dasselbe Modell wie <code>lm()</code> an (durch einen Aufruf von <code>lm</code> für jedes Stratum), aber <code>summary()</code> gibt eine ANOVA-Tabelle aus; nötig für <code>TukeyHSD()</code> und <code>Error()</code> -Strata
<code>coef(model)</code>	Gibt die geschätzten Koeffizienten zurück (teilweise mit ihren Standardfehlern)
<code>summary(model)</code>	Bei <code>lm</code> -Objekten: Tabelle mit Parameterschätzungen, Standardfehlern, t-Statistiken und p-Werten, dazu R^2 , korrigiertes R^2 und den F-Test des Gesamtmodells; bei <code>aov</code> -Objekten: Typ-I-ANOVA-Tabelle mit SS, MS, F-Wert und p-Werten. Kann auch explizit über <code>summary.lm()</code> oder <code>summary.aov()</code> aufgerufen werden
<code>confint(model)</code>	Berechnet Konfidenzintervalle für einen oder mehrere Parameter eines angepassten Modells
<code>residuals(model)</code>	Gibt die Modellresiduen zurück (Kurzform: <code>resid(model)</code>)
<code>fitted(model)</code>	Gibt die angepassten Werte zurück
<code>predict(model)</code>	Gibt Vorhersagen für beobachtete oder erzeugte Werte der Prädiktoren zurück (über das Argument <code>newdata</code>); mit <code>interval = "confidence"</code> oder "prediction" werden Intervalle ergänzt; kann auch explizit über <code>predict.lm()</code> aufgerufen werden

Formel

<code>formula()</code>	Extrahiert Formeln, die in anderen Objekten enthalten sind
<code>formula(y ~ x)</code>	y als Funktion des festen (kontinuierlichen oder kategorialen) x
<code>formula(y ~ Error(x))</code>	Fehlerstrata in <code>aov()</code> , z. B. <code>y ~ A + Error(Subject)</code>
<code>formula(y ~ x1 + x2)</code>	y als Funktion von x1 und x2
<code>formula(y~x1+x2+x1:x2)</code>	y als Funktion von x1 und x2 einschließlich ihrer Interaktion
<code>formula(y ~ x1 * x2)</code>	Kurzschreibweise für die Aufnahme der Interaktion
<code>formula(y~x1+x2%in%x1)</code> oder <code>x1/x2</code> oder <code>x1+x1:x2</code>	y als Funktion von x1 und x2, wobei x2 in x1 genestet ist
<code>! oder x1/x2 oder x1+x1:x2</code> <code>! oder x1/x2 oder x1+x1:x2</code>	y als Funktion von x1 und x2, wobei x2 in x1 genestet ist
<code>formula(y ~ .)</code>	y als Funktion aller anderen Spalten der Daten
<code>formula(y ~ x - 1)</code>	Modell ohne Achsenabschnitt (Intercept)
<code>formula(y ~ I(x^2))</code>	Rechenoperationen in einer Formel müssen in <code>I()</code> stehen; Alternative für Polynome: <code>poly(x, 2)</code>
<code>formula(log(y) ~ x)</code>	Funktionen (z. B. Transformationen) können direkt in der Formel verwendet werden
<code>update(model, ~. -x2)</code>	Aktualisiert Modellformeln, typischerweise durch Hinzufügen oder Entfernen von Termen

Terme testen und Gruppen vergleichen

Diese Funktionen funktionieren gleichermaßen für Regressions-, ANOVA- und ANCOVA-Modelle:

<code>anova(model)</code>	Berechnet die sequenzielle (Typ-I-)Varianztabelle eines angepassten Modells (oder vergleicht mehrere Modelle); die Reihenfolge der Terme in der Formel spielt eine Rolle
<code>car::Anova(model)</code>	Berechnet Typ-II- oder Typ-III-ANOVA-Tabellen für Modellobjekte von <code>aov</code> , <code>lm</code> und weiteren Modellfunktionen; Typ III erfordert Summe-zu-Null-Kontraste
<code>drop1(fit, ~., test="F")</code>	Berechnet marginale SS und F-Tests, indem jeder Term einzeln entfernt wird, unter Beachtung der Marginalität (Haupteffekte, die in einer Interaktion enthalten sind, werden nicht getestet)
<code>oneway.test(y ~ g)</code>	Einfaktorielle ANOVA ohne Annahme gleicher Varianzen (Welch)
<code>TukeyHSD(model)</code>	Berechnet Tukeys Honest Significant Differences für <i>Post-hoc</i> -Vergleiche (für <code>aov</code> -Objekte)
<code>model.tables(model)</code>	Berechnet Übersichtstabellen für Modellanpassungen, insbesondere für komplexe <code>aov</code> -Modelle
<code>plot.design()</code>	Stellt univariate Effekte eines Designs oder Modells dar
<code>interaction.plot()</code>	Interaktionsdiagramm für zwei Faktoren

Faktoren und Kontraste

Standardmäßig verwendet R Treatment-Kontraste: Das erste Level eines Faktors ist die Referenz, und die Koeffizienten sind Unterschiede zu ihr.

<code>relevel(f, ref)</code>	Legt das Referenzlevel eines Faktors fest
<code>contrasts()</code>	Setzt die einem Faktor zugeordneten Kontraste bzw. zeigt sie an
<code>model.matrix()</code>	Erzeugt eine Design- (Modell-)Matrix, z. B. indem Faktoren entsprechend den Kontrasten in Dummy-Variablen zerlegt werden

Modelldiagnostik

Annahmen linearer Modelle: linearer Zusammenhang, unabhängige Fehler, normalverteilte Residuen, gleiche Varianz der Residuen (Homoskedastizität), keine einflussreichen Ausreißer.

<code>plot(model)</code>	Erzeugt Diagnoseplots zur Prüfung der Modellannahmen und einflussreicher Datenpunkte für ein <code>lm</code> -Objekt: Residuen vs. angepasste Werte (Linearität), Q-Q-Plot (Normalverteilung), Scale-Location (gleiche Varianz), Residuen vs. Leverage
<code>cooks.distance(model)</code>	Berechnet Cooks Distanz (Einfluss jeder Beobachtung); große Werte (z. B. > 1) weisen auf einflussreiche Punkte hin
<code>hatvalues(model)</code> <code>rstandard(model)</code> <code>rstudent(model)</code> <code>qqnorm(y)</code>	Leverage jeder Beobachtung Standardisierte Residuen Studentisierte Residuen Erzeugt einen Normal-Quantil-Quantil-Plot der Werte in y
<code>qqplot()</code>	Erzeugt einen Quantil-Quantil-Plot
<code>qqline()</code>	Fügt einem Normal-Quantil-Quantil-Plot (erzeugt mit <code>qqnorm()</code>) eine Linie hinzu, die durch das erste und dritte Quartil verläuft
<code>acf()</code>	Berechnet (und zeichnet standardmäßig) Schätzungen der Autokovarianz- bzw. Autokorrelationsfunktion
<code>pacf()</code>	Berechnet die partielle Autokorrelationsfunktion
<code>ccf()</code>	Berechnet die Kreuzkorrelation bzw. Kreuzkovarianz zweier univariater Zeitreihen
<code>car::vif(model)</code>	Berechnet Varianzinflationsfaktoren zur Erkennung von Kollinearität der Prädiktoren (Werte > 5 deuten auf Probleme hin)

Modellselektion

<code>AIC(model1, model2)</code>	Berechnet das Akaike- Informationskriterium (AIC); je niedriger, desto besser der Kompromiss aus Anpassung und Komplexität (die Modelle müssen an denselben Daten angepasst sein)
<code>BIC(model1, model2)</code>	Berechnet das Bayessche Infor- mationskriterium; bestraft Kom- plexität stärker als das AIC
<code>step(model)</code>	Wählt ein formelbasiertes Mod- ell per AIC in einem schrittweisen Verfahren (Rückwärts- und Vor- wärtsselektion)
<code>anova(m1,m2, test="F")</code>	Vergleicht zwei (oder mehr) gen- estete lm- oder aov-Modelle an- hand des F-Tests
<code>drop1(model, test="F")</code>	Entfernt jeden Term einzeln und vergleicht die Anpassungen (siehe <i>Terme testen</i>)

Tidyverse: die Pakete *broom* und *modelr*

`library(broom)` (wird mit dem tidyverse installiert, aber von `library(tidyverse)` nicht geladen)

Wandelt Modellobjekte (lm, glm, t.test, ...) in Tibbles um:

<code>tidy(model)</code>	Eine Zeile je Modellterm: Schätzwerte, Standard- fehler, Teststatistik und p- Wert; Konfidenzintervalle mit <code>conf.int = TRUE</code>
<code>augment(model)</code>	Eine Zeile je Beobachtung: Beobachtungsdaten plus angepasste Werte, Residuen, Cook's D usw.
<code>augment(model, newdata = nd)</code>	Vorhersagen für neue Daten; mit <code>interval = "prediction"</code> (oder <code>"confidence"</code>) für Inter- valle
<code>glance(model)</code>	Eine Zeile je Modell: Kennzahlen wie R^2 , AIC und BIC

Das Paket *modelr* ist superseded (ein modernes Framework bietet *tidymodels*); `broom::augment()` deckt die meisten Anwendungsfälle ab. Die folgenden Funktionen funktionieren weiterhin:

`modelr::add_predictions(data, model)`

Fügt einem Data Frame die Vorhersagen eines einzelnen Modells hinzu

`modelr::add_residuals(data, model)`

Fügt einem Data Frame die Residuen eines einzelnen Modells hinzu

`modelr::data_grid(data, x1, x2)`

Erzeugt ein gleichmäßiges Gitter von Punkten aus den Daten