Einführung in R

Werkzeuge für das wissenschaftliche Arbeiten

Magnus Bender

11. Januar 2022

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

Allgemeines

Fakten

- R ist ein Software-Paket für statistisches Rechnen.
- R unterstützt eine Vielzahl an Betriebssystemen (Windows, Linux, macOS).
- R ist eine Open Source Implementierung der Sprache S, die in den Bell Laboratories in den 1970er Jahren entwickelt worden ist.
- R wurde ursprünglich von Ross Ihaka und Robert Gentleman an der Universität Auckland Mitte der 1990er Jahre entwickelt.
- R kann über Packages erweitert werden. Einige Packages sind in der R-Distribution enthalten. Weitere befinden sich auf CRAN.

Allgemeines

Ziel der Vorlesung

- Überblick der Möglichkeiten von R
- Einfachen Erstellung von Grafiken mit R und GGPlot2
- Weniger Vermittlung von R als Programmiersprache

Allgemeines

Warum R?

- Python, Julia
- Matlab, SPSS, Mathematica
- "Für Grafiken und Daten kann man ja auch ein Officeprogramm nutzen"

Allgemeines

Quellen – Danke

Diese Slides basieren auf einem Referat von Sascha Frank (Universität Freiburg).

Erweitert und angepasst für diese Vorlesung.

Beispiel I

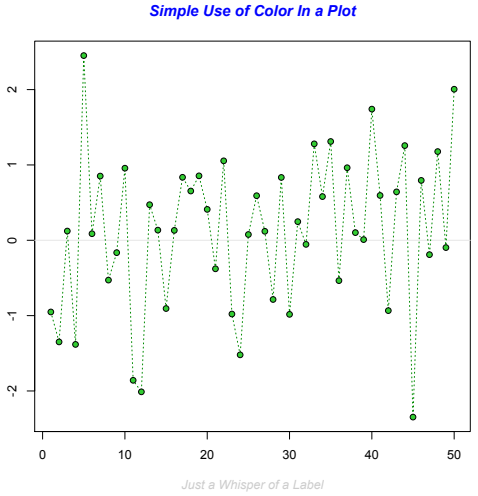
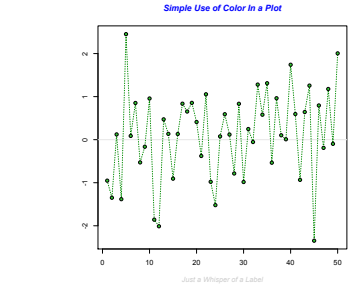
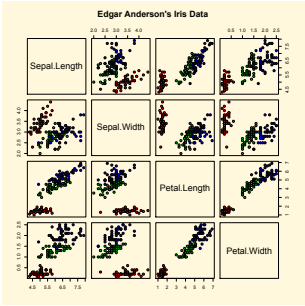
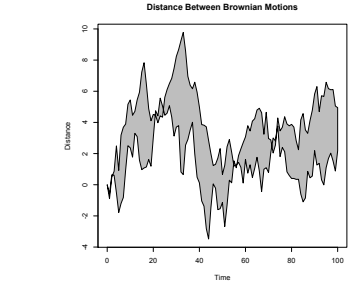


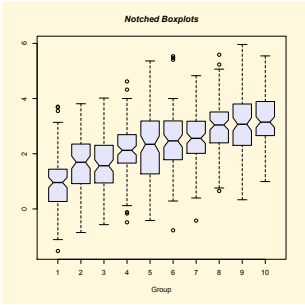
Figure: Plot aus dem Paket graphics

Beispiel II

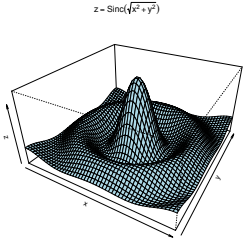

Simple Use of Color In a Plot
Just a Whisper of a Label

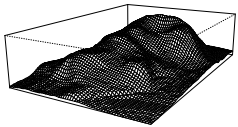

Edgar Anderson's Iris Data
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width

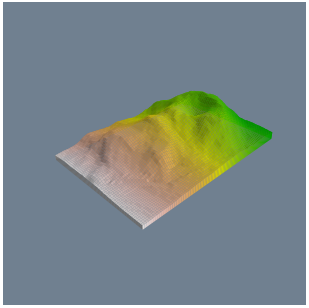

Distance Between Brownian Motions
Distance
Time

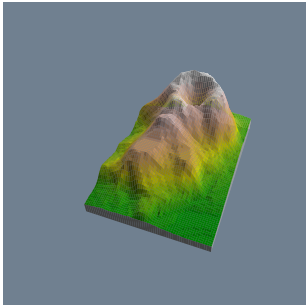

Notched Boxplots
Group

Beispiel III









Vor- und Nachteile

Vorteile

- Open Source
- Aktive Community
- Erweiterbarkeit
- Gut für wissenschaftliches Arbeiten geeignet, z.B. beim Grafikexport

Nachteile

- Keine grafische Benutzeroberfläche
- Keine interaktiven Grafiken
- Eine weitere Programmiersprache

Bezugsquellen

- Informationswebseite: https://r-project.org
- CRAN: Comprehensive R Archive Network
  - Hauptseite: https://cran.r-project.org
- Auf den CRAN Seiten ist R für die unterschiedlichen Distributionen verfügbar

Installation

R

Windows, macOS Download und Ausführung des Installers von CRAN

Linux Über den Paketmanager, z.B. `apt-get install r-base`

- Starten über `Rgui.exe`, `R.app` oder `R` im Terminal

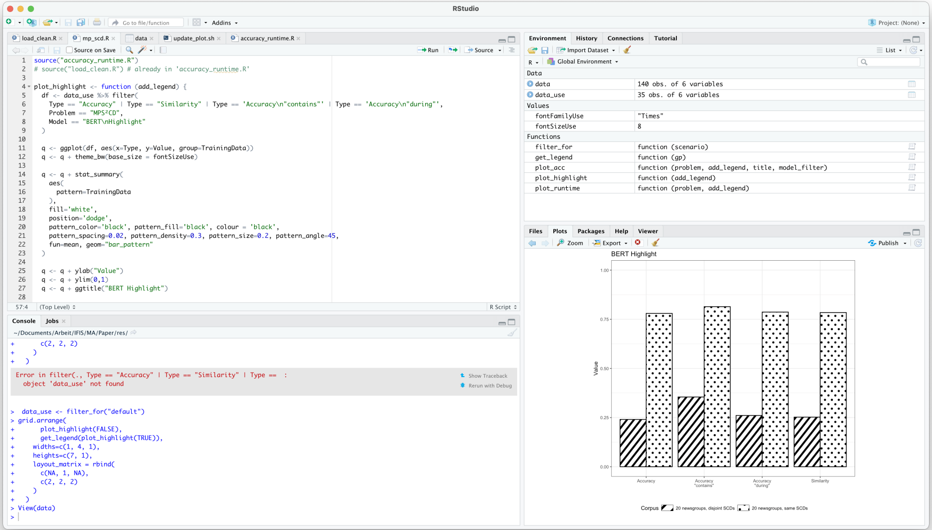
→ Keine GUI, öffnet Fenster "aus dem Terminal"

→ Skripte per `Rscript [--vanilla] <file>.R` ausführen

RStudio

- *Persönliche Empfehlung um R produktiv zu nutzen*
- Open Source Version eines kommerziellen Produkts
- Download und Installation von https://www.rstudio.com/products/rstudio/download/#download
- Bietet eine IDE mit GUI für R

RStudio



RStudio
Go to file/function
Addins
Project: (None)
load_clean.R
mp_scd.R
data
update_plot.sh
accuracy_runtime.R
Source on Save
Run
Source
source("accuracy_runtime.R")
# source("load_clean.R") # already in 'accuracy_runtime.R'
plot_highlight <- function (add_legend) {
df <- data_use %>% filter(
Type == "Accuracy" | Type == "Similarity" | Type == 'Accuracy\n"contains"' | Type == 'Accuracy\n"during"',
Problem == "MPS²CD",
Model == "BERT\nHighlight"
)
q <- ggplot(df, aes(x=Type, y=Value, group=TrainingData))
q <- q + theme_bw(base_size = fontSizeUse)
q <- q + stat_summary(
aes(
pattern=TrainingData
),
fill='white',
position='dodge',
pattern_color='black', pattern_fill='black', colour = 'black',
pattern_spacing=0.02, pattern_density=0.3, pattern_size=0.2, pattern_angle=45,
fun=mean, geom="bar_pattern"
)
q <- q + ylab("Value")
q <- q + ylim(0,1)
q <- q + ggtitle("BERT Highlight")
57:4
(Top Level)
R Script
Console
Jobs
~/Documents/Arbeit/IFIS/MA/Paper/res/
+ c(2, 2, 2)
+)
+)
Error in filter(., Type == "Accuracy" | Type == "Similarity" | Type == :
object 'data_use' not found
Show Traceback
Rerun with Debug
> data_use <- filter_for("default")
> grid.arrange(
+ plot_highlight(FALSE),
+ get_legend(plot_highlight(TRUE)),
+ widths=c(1, 4, 1),
+ heights=c(7, 1),
+ layout_matrix = rbind(
+ c(NA, 1, NA),
+ c(2, 2, 2)
+)
+)
> View(data)
Environment
History
Connections
Tutorial
Import Dataset
List
Global Environment
Data
data
140 obs. of 6 variables
data_use
35 obs. of 6 variables
Values
fontFamilyUse
"Times"
fontSizeUse
8
Functions
filter_for
function (scenario)
get_legend
function (gp)
plot_acc
function (problem, add_legend, title, model_filter)
plot_highlight
function (add_legend)
plot_runtime
function (problem, add_legend)
Files
Plots
Packages
Help
Viewer
Zoom
Export
Publish
BERT Highlight
1.00
0.75
0.50
0.25
0.00
Value
Accuracy
Accuracy "contains"
Accuracy "during"
Similarity
Corpus
20 newsgroups, disjoint SCDs
20 newsgroups, same SCDs

Pakete

- Standardisierte Form der Entwicklung von R
  - *Verpacken* von Funktionen, Sourcen, Datensätzen und Dokumentation
  - Validierungsmöglichkeiten
  - Einfache Verteilung an Dritte, z.B. über CRAN
- Installation mit `install.packages("Paketname")` oder über RStudio
- Laden mit `library(Paketname)`

Nützliche Pakete

`ggplot2`	Grafiken, Plots
`dplyr`	Datenmanipulation
`gridExtra`	Grafiken anordnen
`tikzDevice`	Grafiken als TikZ ausgeben

Informationsquellen

Webseite https://www.r-project.org/

GGPlot2 https://ggplot2.tidyverse.org/

Dplyr https://dplyr.tidyverse.org/

Cheatsheets https://www.rstudio.com/resources/cheatsheets/

Weiteres

- Paketübergreifende Suche in Dokumentationen
  https://www.rdocumentation.org/
- An Introduction in R
- R Data Import/Export

Interaktives Terminal

```
> 1 + 1
    [1] 2
> a <- 0.5
> b <- a * 2
> exp(b)
    [1] 2.718282
```

- Ein Ausdruck in einer Zeile oder mit ; trennen
- Mehrzeilige Ausdrücke im Terminal beginnen in jeder nachfolgenden Zeile mit +
- Einrückung wird ignoriert

Funktionen aufrufen

- Funktionsname getippt wird und danach die Return-Taste

```
> q
    function (save = "default", status = 0, runLast = TRUE)
    .Internal(quit(save, status, runLast))
    <bytecode: 0x7fca0c85e148>
    <environment: namespace:base>
```

→ Code der Funktion `q`

- Funktionsname zusammen mit Argumentenliste innerhalb von `()`

```
> q()
    Save workspace image? [y/n/c]:
```

→ Ausführung der Funktion `q`

Übersicht

Hilfe

```
help(topic), ?topic
?help.search("topic")
?help.start()

> ?exp
```

Operatoren

```
+, -, *, /, ^
&, &&, |, ||
```

Vergleiche

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

Zuweisung

```
x = 5
x <- 5
5 -> x
```

Schleifen

```
for, while, if
```

Kommentare

- Alles nach #

Namen

- Case sensitive
- Buchstaben zu Beginn

Übersicht

Beispiele

Zuweisung oder Vergleich?

`x<-8`

```
> x<-8
> x
   [1] 8
```

`x <- 8`

```
> x <- 8
> x
   [1] 8
```

`x < -8`

```
> x < -8
   [1] FALSE
```

Assignment-Operatoren = und <-

```
> mean(y = 1:10)
    Error in
    mean.default (y = 1:10)
    argument x missing

> mean(y <- 1:10)
    [1] 5.5
```

Datentypen

Elementar

Null	`NULL`
Logical	`TRUE` und `FALSE`
Integer	1, 100, 3236,
Double	1.0, 3.141593, 2.718282, NaN, Inf, $-Inf$
Complex	$1 + 0i$, $1i$, $3 + 5i$
Character	`"Hello"`, `"How are you?"`, `"123"`
Missing value	`NA`

Datentypen

Zusammengesetzt

Vector Elemente desselben elementaren Typs

Array Vektor beliebiger Dimension

Matrix 2-dimensionales Array (Spezialfall, für Matrixoperationen)

factor Spezieller Vektor für das Kodieren von Klassen

list Liste von Elementen mit unterschiedlichen Datentypen (elementar sowie zusammengesetzte)

data.frame *Mischung* aus Matrix und Liste

Zugriffe

- 1-Indexiert
- Dimensionen durch `,` getrennt, z.B. `matrix[1,1]`
- Intervalle mit `:` wählbar, z.B. `matrix[2,3:5]`
- Zeilen oder Spalten vollständig, z.B. `matrix[1,]` oder `matrix[,1]`

Datentypen

Beispiele II

Vector

```
x.vec <- c(10.4, 5.6, 3.1, 6.4, 21.7)
y.vec <- c("a", "b", "c", "d")
z.vec <- c(TRUE, TRUE, TRUE, FALSE)
```

Array

```
x.array <- array(1:24, dim=c(3,4,2))
```

Datentypen

Beispiele II

Data Frame

```
> x <- c(2, 5, 8)
> y <- c("a", "b", "c")
> z <- c(TRUE,TRUE,FALSE)
> df <- data.frame(x,y,z)

> df
     x y     z
   1 2 a  TRUE
   2 5 b  TRUE
   3 8 c FALSE
```

Typumwandlung

- Umwandlung kann mit Hilfe der Funktion `as.<type>` erfolgen
- Prüfen des jeweiligen Typs erfolgt mit Hilfe der Funktion `is.<type>`

```
> x <- matrix(1, nrow=5, ncol=2)
> x
     [,1] [,2]
[1,]    1    1
[2,]    1    1
[3,]    1    1
[4,]    1    1
[5,]    1    1
> is.matrix(x)
[1] TRUE
> is.vector(x)
[1] False
```

```
> y <- as.vector(x)
> y
 [1] 1 1 1 1 1 1 1 1 1 1
```

Datentypen

Listen

```
list <- list(name_1=object_1, ..., name_m=object_m)
union <- c(list_1, list_2, ..., list_n)
```

```
> x <- list(a="Hello", b=1:10, pi=3.1415927)
> x$a
    [1] "Hello"
> x[["b"]]
    [1]  1  2  3  4  5  6  7  8  9 10
> x$pi
    [1] 3.141593
```

```
> length(x)
    [1] 3
> x[[1]]
    [1] "Hello"
> x[1]
    $a
    [1] "Hello"
```

Datentypen

Vektoren

Elemente von Vektoren unterstützen neben Indizes auch Namen

```
> x <- c(5,3,7)
> names(x) <- c("apple", "banana","orange")

> x["apple"]
    apple
        5
> x[1:2] # Zugriff über die Nummer
    apple banana
        5      3
> length(x)
    [1] 3
```

Datentypen

Data Frame

- Können verstanden werden als
  - **Matrix** die in jeder Spalte einen unterschiedlichen Datentyp besitzt
  - **Liste** die in der jedes Element ein Vektor derselben Länge darstellt
- Häufig durch den Import und Datensätzen erzeugt (CSV, Excel, ...)
- Sehr nützlich für die grafische Darstellung der Daten
- Zugriff auf die Elemente analog zu Matrizen oder Listen
- *Missing Values* können mithilfe von `na.omit()` entfernt werden, wodurch die gesamte Zeile gelöscht wird.

```
> df
  x y     z
1 2 a  TRUE
2 5 b  TRUE
3 8 c FALSE
```

Datentypen

Attribute

- Attribute ermöglichen die detaillierte Beschreibung des Inhaltes.
- Ein Array ist z.B. nur ein Vektor mit einem Dimensionsattribut.

```
> df
  x y     z
1 2 a  TRUE
2 5 b  TRUE
3 8 c FALSE
> attributes(df)
$names
[1] "x" "y" "z"
$class
[1] "data.frame"
$row.names
[1] 1 2 3
```

```
> rownames(df) <- c(4,5,6)
> colnames(df) <- c("a", "b", "c")

> df
  a b     c
4 2 a  TRUE
5 5 b  TRUE
6 8 c FALSE
```

Funktionen

```
function_name <- function( name_1 = "default_value", name_2 ) {
    Funktionskörper
}
```

- Schlüsselwort `function` für Funktionsdefinition
- Argumente sind mit Komma getrennte Liste der Form `SYMBOL [= AUSDRUCK]` oder das spezielle Argument `...`
- Jeder gültige Ausdruck kann als `Funktionskörper` der Funktion verwenden werden
- Anonyme Funktionen möglich, ohne `function_name <-`

Funktionen

Erstellen

```
> my_double <- function(x){
+   result <- x*2
+   return(result)}
> my_double(3)
    [1] 9
```

- Ein Rückgabewert (letztes Statement oder Argument von `return`)
- Mehrere Rückgabewerte in Liste zusammenfassen
- Beim Aufruf möglich Namen des Arguments anzugeben `my_double(x=3)`, dann beliebige Reihenfolge
- Bei Definition kann man Default-Werte angeben `function(x=3)`, kann man beim Aufruf weggelassen

Funktionen

Nutzen

```
> set.seed(123456)
> y <- rnorm(100)

> y[1:4]
    [1] 0.83373317 -0.27604777 -0.35500184  0.08748742
> length(y)
    [1] 100

> str(y)
    num [1:100] 0.8337 -0.276 -0.355 0.0875 2.2523 ...
> summary(y)
       Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
   -2.74887 -0.72600  0.04791  0.01682  0.83391  2.50265
```

Funktionen

Das Argument ... I

Beliebig viele Argumente

```
check_num <- function(...) {
    args <- list(...)
    for (i in 1:length(args)) {
        if (!is.numeric(args[[i]])) {
            return(FALSE)
        }
    }
    return(TRUE)}
```

```
> check_num(1, 3)
    [1] TRUE
> check_num(1, "2")
    [1] FALSE
```

Weiterleiten von Argumenten

```
> my_plot <- function(x, y, my_arg, ...) {
    # etwas mit my_arg tun
    plot(x, y, ...)
}
```

Jetzt *versteht* `my_plot()` zusätzlich zu seinen eigenen Argumenten implizit auch alle Argumente von `plot()`.

Funktionen

Lazy Evaluation

Beim Aufruf einer Funktion werden die Argumente nur geparsed, aber nicht evaluiert. Die Auswertung des Argumentes erfolgt erst, wenn das Argument zum ersten mal verwendet wird.

```
my_func <- function(x, y) {
    if(x < 0) return(NaN)
    else return (y * log(x))
}

> my_func(-1)
    [1] NaN
> my_func(2, 3)
    [1] 2.079442
> my_func(2)
    Error in my_func(2) : argument "y" is missing, with no default
```

Nützliche Funktionen

`median(x, na.rm = FALSE)` Median
`na.rm` entscheidet ob *Missing Values* ignoriert werden sollen

`mean(x, trim = 0, na.rm = FALSE)` Arithmetisches Mittel
`trim` gibt an wie viele Elemente am Rand ignoriert werden sollen

`rep(x, times)` Repliziert den Wert `x` `times` mal

`apply(x, MARGIN, FUN)` Führt Funktion `FUN` auf jedes Element von `x` aus
`MARGIN` wählt die Dimension aus

`transform(data, ...)` Spalte in einem Datensatz hinzuzufügen

```
dataFrame <- transform(dataFrame,
                       newColumn = oldColumn1 + oldColumn2)
```

Flusskontrolle

- Als vollwertige Programmiersprache besitzt R eine Reihe von Strukturen zur Kontrolle des Programmflusses.
- Iterationen werden durch die Befehle `for`, `while` und `repeat` zur Verfügung gestellt.
- Für bedingte Auswertungen können `if`, `else` verwendet werden

If-Else

Ein typisches Beispiel für die Verwendung von `if`:

```
if (any (x < 0))
    stop("negative Werte in x")
```

Auswahl zwischen mehreren Möglichkeiten:

```
if (all (x >= 0)) {
    sqrt(x)
} else { # optional
    sqrt(x + 0i)
}
```

Einlesen von Daten

In R gibt es die Möglichkeit, die Daten aus den verschiedensten Formaten einzulesen.
Häufig benötigte Befehle sind:

`read.table` Einlesen von Daten in Tabellen-Format aus einer Datei und erzeugen eines Data Frames. Wichtige Parameter sind z.B. `header`, `sep`.

`read.csv()` **und** `read.csv2()` Funktionen für Dateien im CSV-Format (z.B. Excel und andere Exporte). Bei `read.csv2()`sind die Einträge durch Semikola getrennt und das Dezimalzeichen ist – wie im Deutschen üblich – ein Komma

`load()` **und** `save()` Natives laden und speichern von Daten (und auch Objekten) durch R

Auflistungen

Wichtige Funktionen

`q()`	Beende die Session. Es wird gefragt, ob der Workspace gespeichert werden soll
`help()`	Bekomme Hilfeseite für eine Funktion oder ein Objekt
`help.start()`	Verwende einen Webbrowser für das Lesen der Hilfe
`ls()`	Liste die Objekte im Workspace auf
`rm()`	Lösche Objekte im Workspace
`save.image()`	Speichern des Workspace-Inhalts in einem File
`save()`	Speichern eines Objekts in einem File
`load()`	Laden von mit R gespeicherten Daten
`example("as.array")`	Führe das Beispiel auf der Hilfeseite aus
`str()`	Anzeigen der Struktur eines Objekts
`summary()`	Fasse die Information über ein Objekt kurz zusammen
`plot()`	Erstelle eine Graphik für ein Objekt

Auflistungen

Arithmetische Operatoren

Operator	Beschreibung
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

Auflistungen

Logische Operatoren

Operator	Beschreibung
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if x is TRUE

GGPlot2

Ein Paket zur Erstellung von Grafiken mit R

GGplot2

- Wendet *Grammar of Graphics* an
- Grafiken deklarativ erstellen
- Trennung in semantische Komponenten

Statistische Grafik

"In brief, the grammar tells us that a statistical graphic is a mapping from data to aesthetic attributes (colour, shape, size) of geometric objects (points, lines, bars). The plot may also contain statistical transformations of the data and is drawn on a specific coordinate system."
– Hadley Wickham (developer of GGPlot2)[1]

[1] https://ggplot2-book.org/introduction.html

Begrifflichkeiten

Data Daten die visualisiert werden sollen

Coord Koordinatensystem, in dem die Daten visualisiert werden sollen

Geoms Geometrische Objekte die für die Repräsentation von Daten verwendet werden (Punkte, Linien, Histogramme, Boxplots, ...)

Mappings Zuordnung zwischen Variablen der Daten und ästhetischen Attributen der geometrischen Objekte

Scales Abbildung der Merkmalsausprägungen auf Einheiten der geometrischen Attribute (Form, Durchmesser, Farbe)

Stats Statistische Transformation der Daten (z.B. Mittelwert, Standardabweichung, ...)

Beispiele

Daten

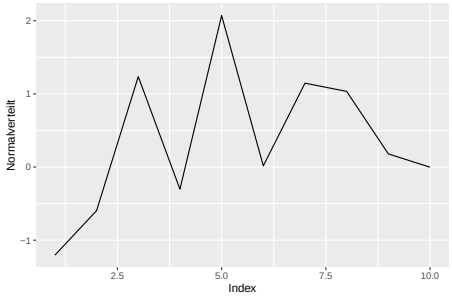
	Index	Normalverteilt	Gruppe
1	1	-1.205329072	a
2	2	-0.595916414	a
3	3	1.236501856	a
4	4	-0.302331151	a
5	5	2.071155984	a
6	6	0.016901578	b
7	7	1.147090110	b
8	8	1.034211873	b
9	9	0.179584737	b
10	10	-0.002140626	b

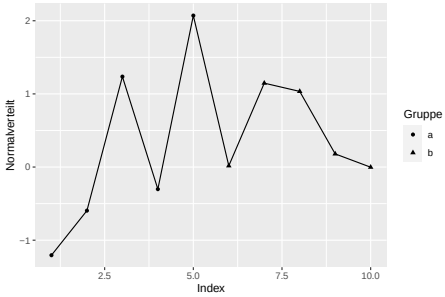
```
library(ggplot2)

x <- 1:10
y <- rnorm(10)
g <- c("a", "a", "a", "a", "a", "b", "b", "b", "b", "b")
data <- data.frame(x,y,g)

colnames(data) <- c("Index", "Normalverteilt", "Gruppe")
```

Beispiele

Linie und Punkte

```
p <- ggplot(data, aes(x=Index, y=Normalverteilt))
p <- p + geom_line()
p # linkes Bild
p <- p + geom_point(aes(shape=Gruppe))
p # rechtes Bild
```

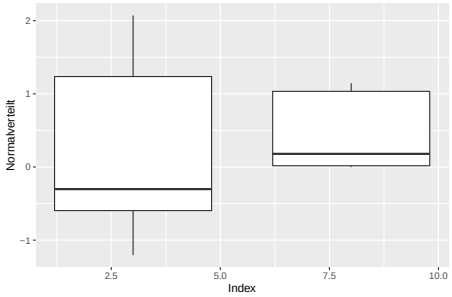




Beispiele

Boxplot

```
p <- ggplot(data, aes(x=Index, y=Normalverteilt))
p <- p + geom_boxplot(aes(group=Gruppe))
p
```

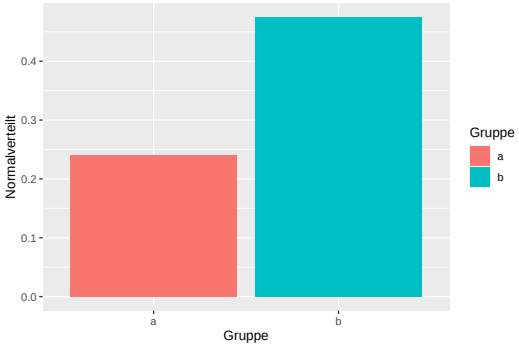


Sinnvolle Darstellung dieser Daten?

Beispiele

Transformationen

```
p <- ggplot(data, aes(x=Gruppe, y=Normalverteilt, fill=Gruppe))
p <- p + stat_summary(position="dodge", fun=mean, geom="bar")
p
```



Grundlagen

Daten und Mapping

```
ggplot(data, mapping = aes(), ...)
```

`data` Der zu visualisierende Data Frame

`mapping` Liste von ästhetischen Zuordnungen

- Können auch erst bei der Darstellung gewählt werden
- Können nach Reihenfolge oder mit Parameternamen übergeben werden
- Nutzung der Spaltennamen des Data Frame

Grundlagen

Darstellung und Geoms

```
geom_XXX(mapping = NULL, data = NULL, stat, ...)
```

- Auswahl der Darstellung
- Erneut können Daten und Mappings angegeben werden
- Unterschiedliche Parameter je Darstellung (z.B. Farbe, Linientyp, Form, ...)

Grafik

- Die Geoms werden durch Addition einem GGPlot-Objekt hinzugefügt
- Verschiedene weitere GGPlot-Objekte können hinzuaddiert werden
- Erst durch Ausgabe des GGPlot-Objektes wird der Plot erzeugt

→ **Vorstellung eines etwas größeren Plots nächste Woche**

Projektaufgabe

- Bestehen des Vorlesungsteils R durch Bearbeiten der Projektaufgabe
- Aufgabenstellung und Datensatz im Moodle
- R, RStudio und die hier genannten Pakete dürfen genutzt werden
- Gruppen von (maximal) drei Personen (Namen in der Abgabe angeben!)
- Jede*r Studierende muss seine erarbeitete Lösung eigenständig im Moodle hochladen
- R-Quellcode sowie Grafiken als PDF oder PNG im Moodle hochladen

Projektaufgabe

Zeitplan

Heute **(i)** Vorlesung
(ii) Freischaltung Projektaufgabe

18.01. **(i)** Vorstellung eines etwas größeren Plots mit GGPlot2
(ii) Übung für Fragen zur Vorlesung und Aufgabenstellung

24.01. Abgabe Projektaufgabe

Bis nächste Woche!

Magnus Bender

bender@ifis.uni-luebeck.de