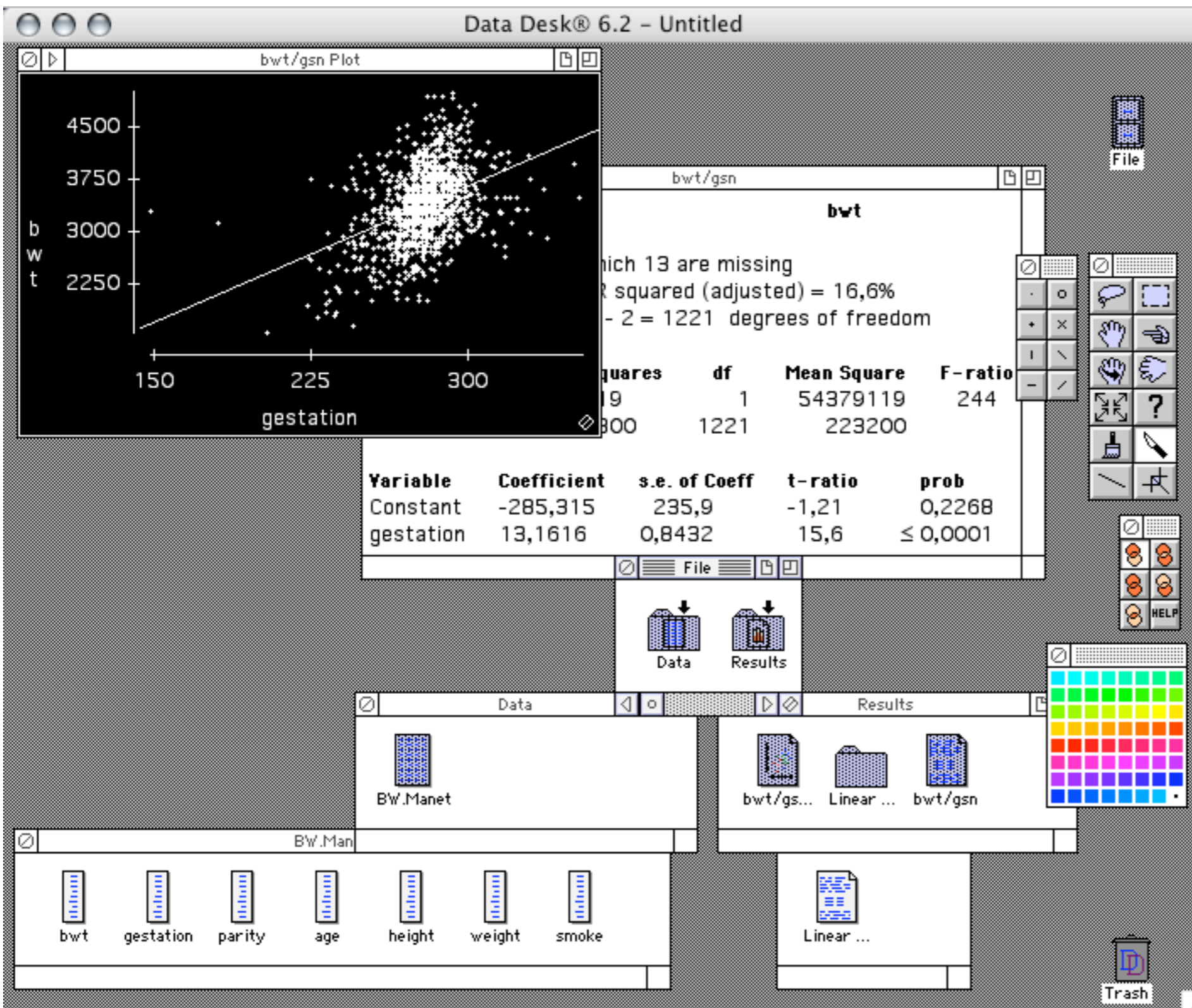
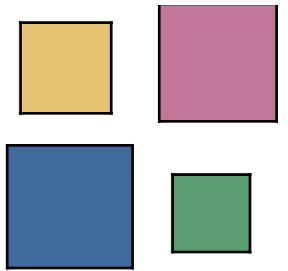


EDA: Arbeitsumgebungen

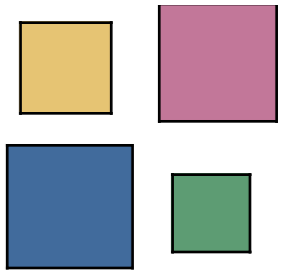


- **Data Desk**
 - Daten (relational)
 - Ergebnisse
 - Hierarchisch in Ordnern
 - Alle Objekte bleiben erhalten
- **Vorteil**
 - Hohe Transparenz
- **Nachteil**
 - Schlechte Reproduzierbarkeit



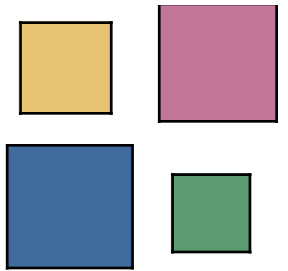
R und EDA: Workspaces

- Jedes Objekt, auf das R zugreifen können soll, wird im sog. “Workspace” von R verwaltet.
 - Alle Daten mit denen R arbeitet, werden in den Workspace geladen (d.h. was nicht in den Hauptspeicher passt, kann nicht bearbeitet werden!)
 - Der Standard Workspace wird in der Datei `.RData` im Home-Verzeichniss des Benutzers angelegt
 - Startet man R aus der Shell (Eingabeaufforderung), arbeitet R in dem Verzeichnis, wo man es startet, mit der Datei `.RData`.
 - Der Workspace wird nicht automatisch gesichert, sondern muss/kann explizit vom Benutzer mit den Kommandos `save.image(...)` und `load(...)` gespeichert, bzw. geladen werden.
 - Mögliche Arbeitsweise:
Für jedes Projekt/Datensatz verwendet man einen eigenen Workspace



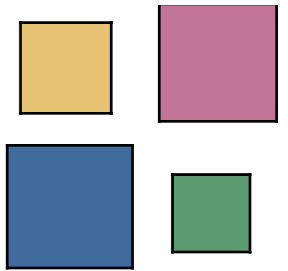
R und EDA: Data Frames

- Die primären Datentypen in R sind:
 - **Skalare** (`1.234`, `pi`, `T`, ...)
 - **Vektoren** (`c(1,4,7)`, `1:10`, ...)
 - **Matrizen** (`matrix(0, 5, 5)`, `diag(4)`)
 - **Listen** (`list(1, 1:10, "Hello World", c("a", "b", "c"))`)
- Data Frames stellen Datensätze in R dar, daher können sie nur in Form einer Liste erstellt werden, um nicht auf nur numerische Objekte beschränkt zu sein.
- Die Kommandos `attach(...)` und `detach(...)` erleichtern den Zugriff auf Komponenten eines Data Frames (Achtung bei Zuweisung auf "attachte" Objekte!)
- Im Gegensatz zu Listen können Data Frames wie eine Matrix indiziert werden.



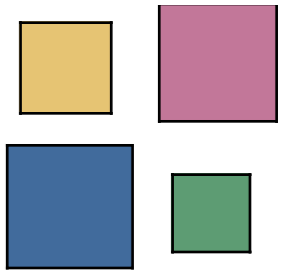
R und EDA: Datentypen in Data Frames

- Prinzipiell unterscheidet man drei Skalenniveaus:
 - stetig
 - ordinal
 - diskret
- Bei der Speicherung der Daten ergeben sich zwei Typen:
 - numerisch
 - alpha-numerisch
- Werden Daten in R geladen (`read.table(...)`), so gilt:
 - numerisch \Rightarrow stetig \Rightarrow `numeric`
 - alpha-numerisch \Rightarrow diskret \Rightarrow `factor`
- R erkennt keine numerischen Faktoren!
- Achtung:
Je nach Funktion **MUSS** u.U. eine Eingabe ein Faktor sein
(z.B. `rpart` berechnet je nach Typ der Zielvariablen einen Klassifikationsbaum oder einen Regressionsbaum.)



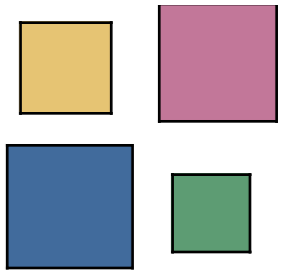
R und EDA: Faktoren

- Faktoren können numerisch und alpha-numerisch sein.
- Alpha-numerische Faktoren werden intern auch wieder über Zahlen, d.h. Integer repräsentiert.
- Um auf die Stufen eines Faktors zuzugreifen, benötigt man die Funktion `unclass(...)`.
- Die Stufen eines Faktors werden mit der Funktion `levels(...)` aufgelistet.
- Existiert eine Ordnung auf den Stufen des Faktors, kann dies mittels der Funktion `ordered(...)` definiert werden.



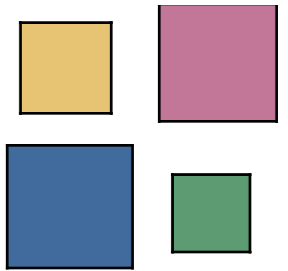
R und EDA: Objektverwaltung

- Auflisten aller Objekte: `ls (...)`
 - mit dem optionalen Parameter `pattern="..."` kann man gezielt nach Objekten suchen (via regulären Ausdrücken)
 - der Parameter `pos=...` erlaubt es jegliches Objekt in R zu listen (Standardmäßig werden nur die User-Objekte gezeigt, aber keine Systemkomponenten)
- Typen von Objekten:
 - `class (...)`
liefert den abstrakten Typ eines Objektes
(z.B. ist ein Data Frame von der Klasse "`data.frame`" aber auch gleichzeitig vom Typ "`list`")
 - `mode (...)`
liefert den logischen Typ eines Objektes
 - `typeof (...)`
liefert den "physikalischen" Speichertyp eines Objektes



R und EDA: Benutzer Interface

- R basiert auf dem REPL Prinzip, d.h. auf einer **Read, Evaluate, Print Loop**.
- \Rightarrow während R rechnet, ist keine Kommunikation mit R möglich!
- Die R-spezifische Benutzer Schnittstelle ist das Terminal!
- Alle anderen Varianten, d.h. GUIs (**G**raphical **U**ser **I**nterfaces), Editoren, Tabellen und graphische Ausgabefenster sind systemspezifisch.
- Große Heterogenität zwischen den verschiedenen Implementationen (gerade in der Lehre sehr unangenehm)
- Keines der GUIs unterstützt spezifisch die Analyse von Daten
 \Rightarrow **JGR** (Java Gui for R).

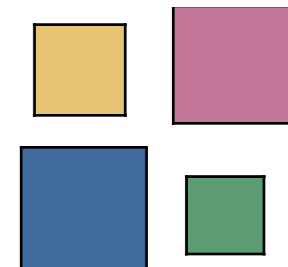


R und EDA: Graphik

- Das grundlegende Graphik System in R wird bald 40 ...
(basiert auf dem “*pen on paper*” Prinzip)
- Graphische Ausgaben werden immer auf ein Device geleitet.

Typische Devices:

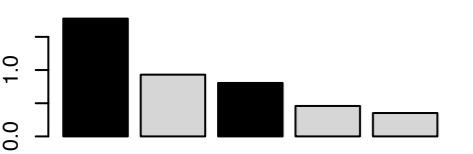
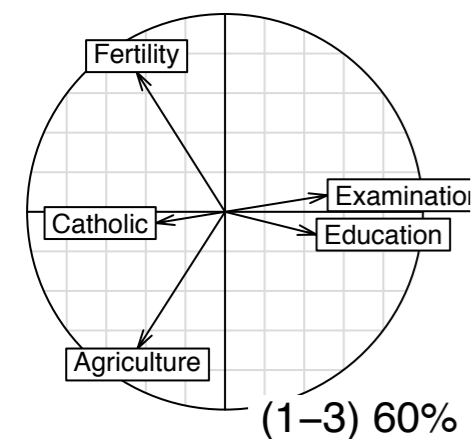
- Native Bildschirmausgabe
 - `X11()`
 - `windows()`
 - `quartz()`
- `postscript()`
- `pdf()`
- `png()`
- `jpeg()`
- `pictex()`
- `JavaGD()` ...
- Die einzigen Kommandos zur Benutzer Interaktion sind `locator(...)` und `identify(...)` ⇒ **iPlots**



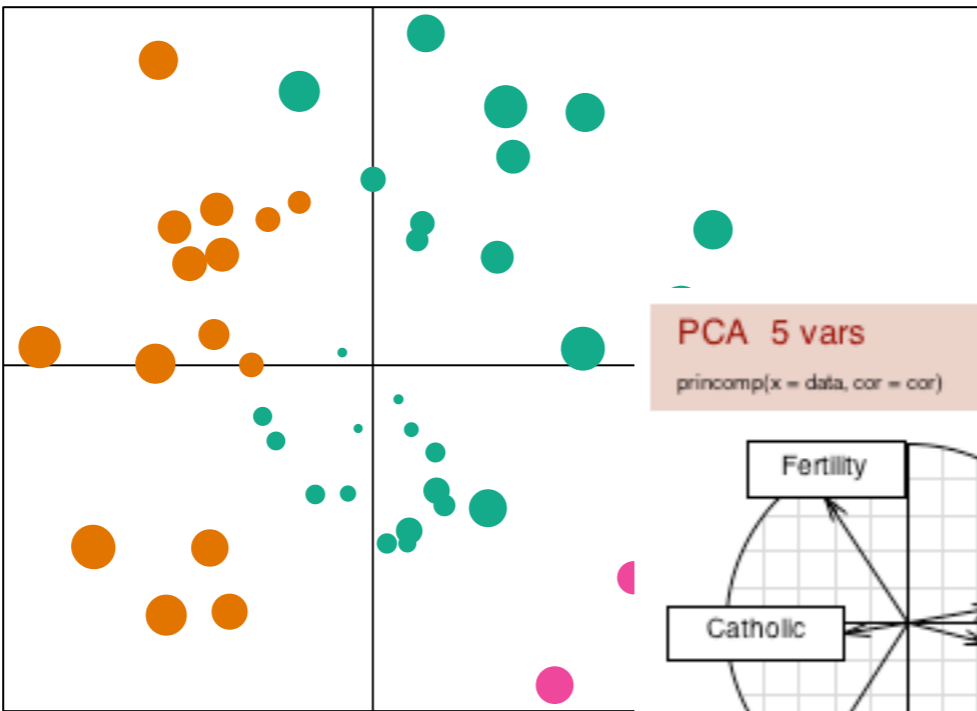
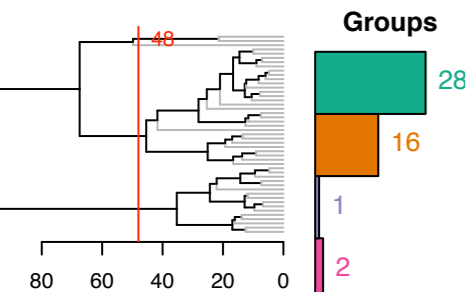
Graphische Stabilität ...

- pdf

PCA 5 vars
princomp(x = data, cor = cor)

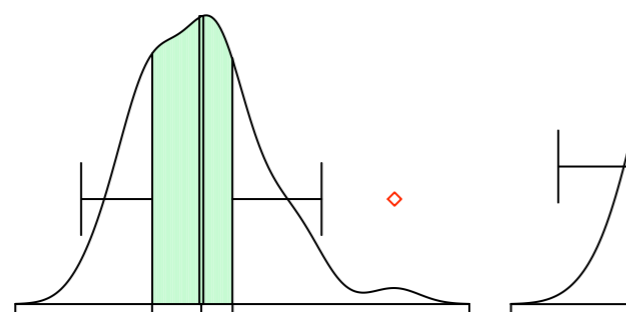


Clustering 4 groups



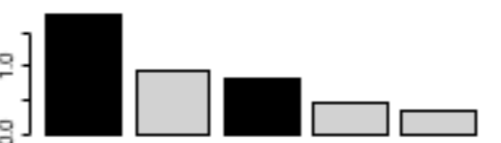
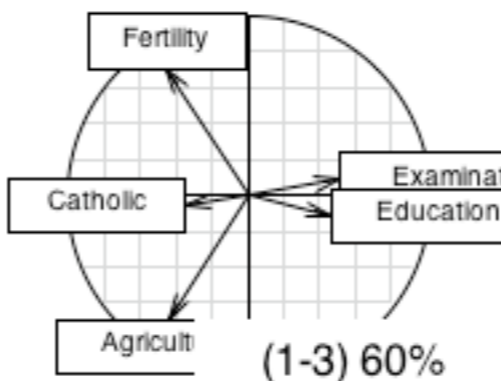
Factor 1 [41%]

Factor

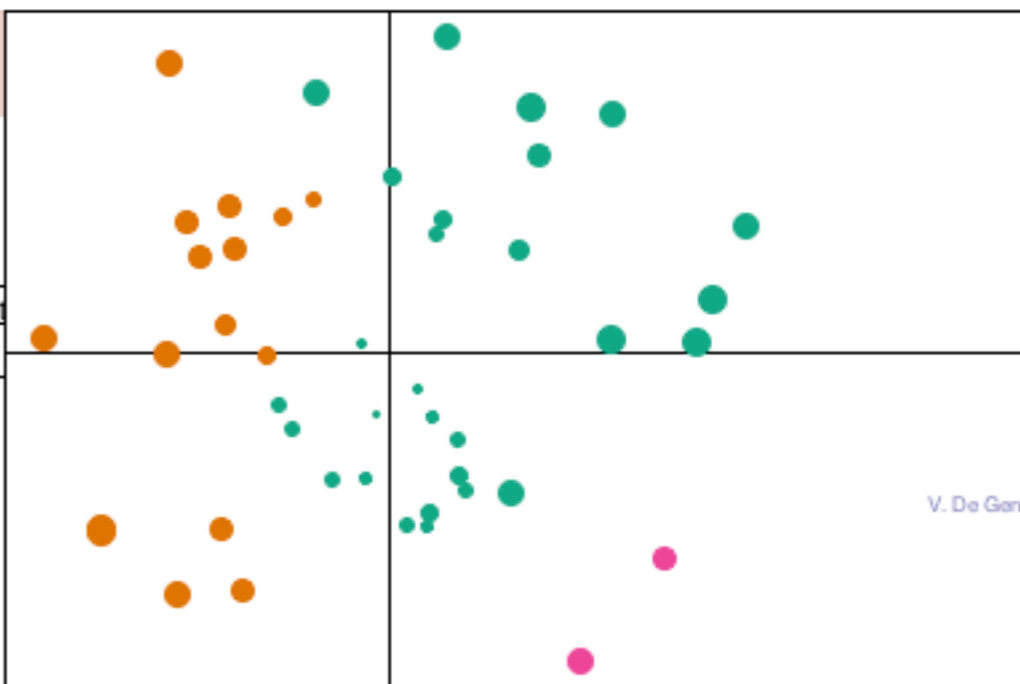
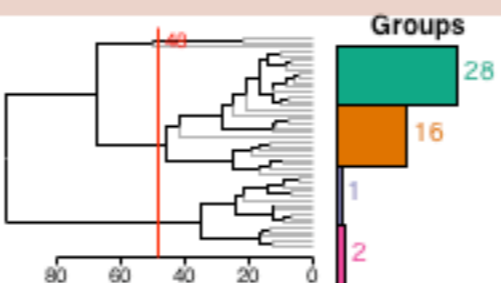


- JavaGD

PCA 5 vars
princomp(x = data, cor = cor)



Clustering 4 groups

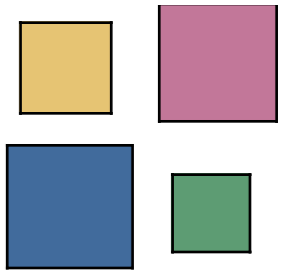


Factor 1 [41%]

Factor 3 [19%]



- ACHTUNG!!!



R Graphik: Grid und Lattice

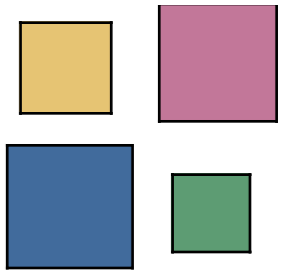
- **Grid**

Grid Graphics ist ein moderneres low-level Graphik System, was sich mehr an Postscript orientiert und mehr Optionen bietet als das alte System

- **Lattice**

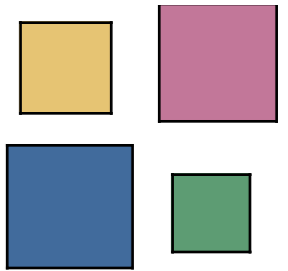
Lattice Graphics implementiert Trellis Graphics, und soll ein abstrahierendes Font-End zu der Grid Graphics sein, zur einfacheren Erstellung von Graphiken in der Datenanalyse.

- Oft existieren daher zwei “Inkarnationen” von Plots, z.B. `parcoord(...)` und `parallel(...)`.
- Lattice Graphics verwendet die R Formelsprache, ist sehr flexibel, aber oft zu kompliziert.



R und EDA: Missing Values

- Fehlende Werte werden standardmäßig als `'NA'` erwartet.
- `read.table(...)` kann aber auch andere Marker für fehlende Werte verwenden (leere Einträge sind auch Missings)
- Viele Funktionen in R behandeln Missings “automatisch”, die Standardoption ist dabei `na.rm=T`, d.h. Missings werden ignoriert.
- Aggregationsfunktionen geben als Ergebnis meist `'NA'` zurück, sobald ein fehlender Wert in der Eingabe vorhanden ist.
- **Fazit**
Die konkrete Behandlung von Missings sollte immer überprüft werden, da es kein garantiertes Standardverhalten gibt.



R und EDA: Klassische EDA-Funktionen

- **`fivenum(...)`**
Five number Summary
- **`stem(...)`**
stem & leaf Plot
- **`rootogram(...)`**
im Paket “vcd”
- **`tmd(...)`**
mean-difference Plot im Paket “lattice”
- **`smooth(...)`**
Tukey's smoothers, 3RS3R, 3RSS, 3R, etc
- ...