

# Praxis der Datenanalyse

Skript zum Modul

*Sebastian Sauer, Matthias Gehrke, Karsten Lübke, Oliver Gansser*

*22 April, 2017*



# Inhaltsverzeichnis

1	Organisatorisches . . . . .	vii
2	Rahmen . . . . .	vii
3	Datenjudo . . . . .	vii
3.1	Typische Probleme . . . . .	ix
3.2	Daten aufbereiten mit <code>dplyr</code> . . . . .	ix
3.3	Die Pfeife . . . . .	xxv
3.4	Befehlsübersicht . . . . .	xxx
3.5	Verweise . . . . .	xxx
4	Praxisprobleme der Datenaufbereitung . . . . .	xxxi
5	Fallstudie zum Datenjudo . . . . .	xxxi
6	Daten visualisieren . . . . .	xxxi
7	Fallstudie zur Visualisierung . . . . .	xxxi
8	Grundlagen des Modellierens . . . . .	xxxi
9	Der p-Wert . . . . .	xxxi
10	Klassische lineare (numerische) Regression . . . . .	xxxi
11	Klassifizierende Regression . . . . .	xxxi
12	Fallstudien zum geleiteten Modellieren . . . . .	xxxi
13	nicht robust: . . . . .	xxxi
14	Vertiefung: Clusteranalyse . . . . .	xxxi
15	Probeklausur . . . . .	xxxi
16	Literaturverzeichnis . . . . .	xxxi



# Tabellenverzeichnis



# Abbildungsverzeichnis

1	Daten aufbereiten . . . . .	viii
2	Lego-Prinzip: Zerlege eine komplexe Struktur in einfache Bausteine . . . . .	x
3	Durchpfeifen: Ein Dataframe wird von Operation zu Operation weitergereicht	x
4	Zeilen filtern . . . . .	xi
5	Spalten auswählen . . . . .	xiii
6	Spalten sortieren . . . . .	xvi
7	Datensätze nach Subgruppen aufteilen . . . . .	xvii
8	Schematische Darstellung des 'Gruppieren - Zusammenfassen - Kombinieren'	xix
9	Spalten zu einer Zahl zusammenfassen . . . . .	xx
10	La trahison des images [Ceci n'est pas une pipe], René Magritte, 1929, © C. Herscovici, Brussels / Artists Rights Society (ARS), New York, <a href="http://collections.lacma.org/node/239578">http://collections.lacma.org/node/239578</a> . . . . .	xxvi
11	Das 'Durchpfeifen' . . . . .	xxvi
12	Sinnbild für mutate . . . . .	xxix

## Vorwort

### 1 Organisatorisches

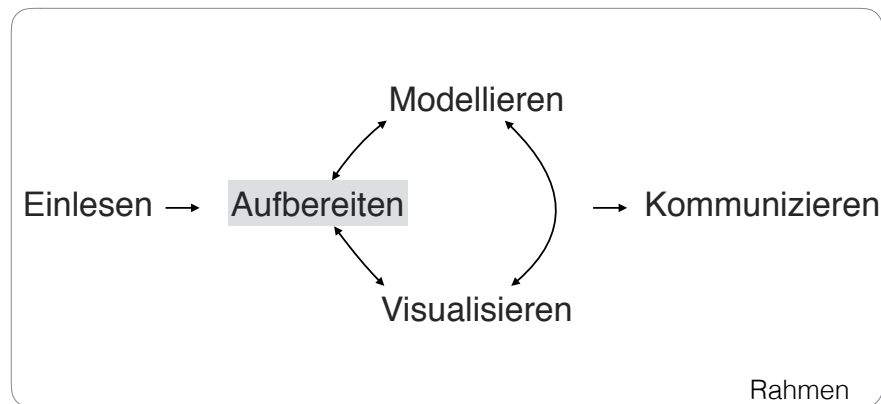
### 2 Rahmen

### 3 Datenjudo



#### Lernziele:

- Typische Probleme der Datenanalyse schildern können.
- Zentrale `dplyr`-Befehle anwenden können.
- `dplyr`-Befehle kombinieren können.
- Die Pfeife anwenden können.
- Werte umkodieren und “binnen” können.



**Abbildung 1:** Daten aufbereiten

In diesem Kapitel benötigte Pakete:

```
library(tidyverse) # Datenjudo
library(stringr)   # Texte bearbeiten
library(car)       # für 'recode'
```

Das Paket `tidyverse` lädt `dplyr`, `ggplot2` und weitere Pakete<sup>1</sup>. Daher ist es komfortabler, `tidyverse` zu laden, damit spart man sich Tipparbeit. Die eigentliche Funktionalität, die wir in diesem Kapitel nutzen, kommt aus dem Paket `dplyr`.

Mit *Datenjudo* ist gemeint, die Daten für die eigentliche Analyse “aufzubereiten”. Unter *Aufbereiten* ist hier das Umformen, Prüfen, Bereinigen, Gruppieren und Zusammenfassen von Daten gemeint. Die deskriptive Statistik fällt unter die Rubrik *Aufbereiten*. Kurz gesagt: Alles, was man tut, nachdem die Daten “da” sind und bevor man mit anspruchsvoller(er) Modellierung beginnt.

Ist das *Aufbereiten* von Daten auch nicht statistisch anspruchsvoll, so ist es trotzdem von großer Bedeutung und häufig recht zeitintensiv. Eine Anekdote zur Relevanz der Datenaufbereitung, die (so will es die Geschichte) mir an einer Bar nach einer einschlägigen Konferenz erzählt wurde (daher keine Quellenangabe, Sie verstehen. . .). Eine Computerwissenschaftlerin aus den USA (deutschen Ursprungs) hatte einen beeindruckenden “Track Record” an Siegen in Wettkämpfen der Datenanalyse. Tatsächlich hatte sie keine besonderen, raffinierten Modellierungstechniken eingesetzt; klassische Regression war ihre Methode der Wahl. Bei einem Wettkampf, bei dem es darum ging, Krebsfälle aus Krankendaten vorherzusagen (z.B. von Röntgenbildern) fand sie nach langem Datenjudo heraus, dass in die “ID-Variablen” Information gesickert war, die dort nicht hingehörte und die sie nutzen konnte für überraschend (aus Sicht der Mitstreiter) gute Vorhersagen zu Krebsfällen. Wie war das möglich? Die Daten stammten aus mehreren Kliniken, jede Klinik verwendete ein anderes System, um IDs für Patienten zu erstellen. Überall waren die IDs stark genug, um die Anonymität der Patienten sicherzustellen, aber gleichwohl konnte man (nach einigem Judo) unterscheiden, welche ID

<sup>1</sup>für eine Liste s. `tidyverse_packages(include_self = TRUE)`



von welcher Klinik stammte. Was das bringt? Einige Kliniken waren reine Screening-Zentren, die die Normalbevölkerung versorgte. Dort sind wenig Krebsfälle zu erwarten. Andere Kliniken jedoch waren Onkologie-Zentren für bereits bekannte Patienten oder für Patienten mit besonderer Risikolage. Wenig überraschen, dass man dann höhere Krebsraten vorhersagen kann. Eigentlich ganz einfach; besondere Mathe steht hier (zumindest in dieser Geschichte) nicht dahinter. Und, wenn man den Trick kennt, ganz einfach. Aber wie so oft ist es nicht leicht, den Trick zu finden. Sorgfältiges Datenjudo hat hier den Schlüssel zum Erfolg gebracht.

### 3.1 Typische Probleme

Bevor man seine Statistik-Trickkiste so richtig schön aufmachen kann, muss man die Daten häufig erst noch in Form bringen. Das ist nicht schwierig in dem Sinne, dass es um komplizierte Mathe ginge. Allerdings braucht es mitunter recht viel Zeit und ein paar (oder viele) handwerkliche Tricks sind hilfreich. Hier soll das folgende Kapitel helfen.

Typische Probleme, die immer wieder auftreten, sind:

- *Fehlende Werte*: Irgend jemand hat auf eine meiner schönen Fragen in der Umfrage nicht geantwortet!
- *Unerwartete Daten*: Auf die Frage, wie viele Facebook-Freunde er oder sie habe, schrieb die Person “I like you a lot”. Was tun???
- *Daten müssen umgeformt werden*: Für jede der beiden Gruppen seiner Studie hat Joachim einen Google-Forms-Fragebogen aufgesetzt. Jetzt hat er zwei Tabellen, die er “verheiraten” möchte. Geht das?
- *Neue Variablen (Spalten) berechnen*: Ein Student fragt nach der Anzahl der richtigen Aufgaben in der Statistik-Probeklausur. Wir wollen helfen und im entsprechenden Datensatz eine Spalte erzeugen, in der pro Person die Anzahl der richtig beantworteten Fragen steht.

### 3.2 Daten aufbereiten mit dplyr

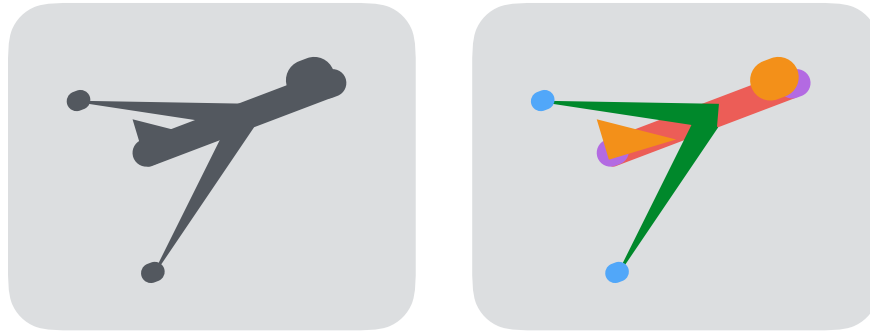
Es gibt viele Möglichkeiten, Daten mit R aufzubereiten; **dplyr**<sup>2</sup> ist ein populäres Paket dafür. **dplyr** basiert auf zwei Ideen:

1. “Lego-Prinzip”: Komplexe Datenanalysen in Bausteine zerlegen (vgl. Abb. 2).
2. “Durchpfeifen”: Alle Operationen werden nur auf Dataframes angewendet; jede Operation erwartet einen Dataframe als Eingabe und gibt wieder einen Dataframe aus (vgl. Abb. 3).

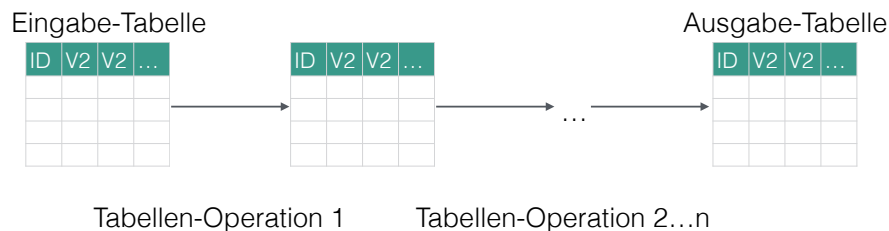
Eine zentrale Idee von **dplyr** ist, dass es nur ein paar wenige Grundbausteine geben sollte, die sich gut kombinieren lassen. Sprich: Wenige grundlegende Funktionen mit eng umgrenzter Funktionalität. Der Autor, Hadley Wickham, sprach einmal in einem Forum (citation needed), dass diese Befehle wenig können, das Wenige aber gut. Ein Nachteil dieser Konzeption

---

<sup>2</sup><https://cran.r-project.org/web/packages/dplyr/index.html>



**Abbildung 2:** Lego-Prinzip: Zerlege eine komplexe Struktur in einfache Bausteine



**Abbildung 3:** Durchpfeifen: Ein Dataframe wird von Operation zu Operation weitergereicht

kann sein, dass man recht viele dieser Bausteine kombinieren muss, um zum gewünschten Ergebnis zu kommen. Außerdem muss man die Logik des Baukastens gut verstanden haben - die Lernkurve ist also erstmal steiler. Dafür ist man dann nicht darauf angewiesen, dass es irgendwo “Mrs Right” gibt, die genau das kann, was ich will. Außerdem braucht man sich auch nicht viele Funktionen merken. Es reicht einen kleinen Satz an Funktionen zu kennen (die praktischerweise konsistent in Syntax und Methodik sind).

Willkommen in der Welt von **dplyr**! **dplyr** hat seinen Namen, weil es sich ausschließlich um *Dataframes* bemüht; es erwartet einen Dataframe als Eingabe und gibt einen Dataframe zurück (zumindest bei den meisten Befehlen).

Diese Bausteine sind typische Tätigkeiten im Umgang mit Daten; nichts Überraschendes. Schauen wir uns diese Bausteine näher an.

### 3.2.1 Zeilen filtern mit **filter**

Häufig will man bestimmte Zeilen aus einer Tabelle filtern; **filter**. Zum Beispiel man arbeitet für die Zigarettenindustrie und ist nur an den Rauchern interessiert (die im Übrigen unser Gesundheitssystem retten (Krämer 2011)), nicht an Nicht-Rauchern; es sollen die nur Umsatzzahlen des letzten Quartals untersucht werden, nicht die vorherigen Quartale; es sollen nur die Daten aus Labor X (nicht Labor Y) ausgewertet werden etc.

Abb. 4 zeigt ein Sinnbild für **filter**.

Merke:

Die Funktion **filter** filtert Zeilen aus einem Dataframe.

ID	Name	Note1
1	Anna	1
2	Anna	1
3	Berta	2
4	Carla	2
5	Carla	2

→

ID	Name	Note1
1	Anna	1
2	Anna	1

Abbildung 4: Zeilen filtern

Schauen wir uns einige Beispiel an; zuerst die Daten laden nicht vergessen. Achtung: “Wohnen” die Daten in einem Paket, muss dieses Paket installiert sein, damit man auf die Daten zugreifen kann.

```
data(profiles, package = "okcupiddata") # Das Paket muss installiert sein
```

```
df_frauen <- filter(profiles, sex == "f") # nur die Frauen
df_alt <- filter(profiles, age > 70) # nur die alten Menschen
df_alte_frauen <- filter(profiles, age > 70, sex == "f")
# nur die alten Frauen, d.h. UND-Verknüpfung

df_nosmoke_nodrinks <- filter(profiles, smokes == "no" | drinks == "not at all")
# liefert alle Personen, die Nicht-Raucher *oder* Nicht-Trinker sind
```

Gar nicht so schwer, oder? Allgemeiner gesprochen werden diejenigen Zeilen gefiltert (also behalten bzw. zurückgeliefert), für die das Filterkriterium TRUE ist.



Manche Befehle wie `filter` haben einen Allerweltsnamen; gut möglich, dass ein Befehl mit gleichem Namen in einem anderen (geladenen) Paket existiert. Das kann dann zu Verwirrungen führen - und kryptischen Fehlern. Im Zweifel den Namen des richtigen Pakets ergänzen, und zwar zum Beispiel so: `dplyr::filter(...)`.

### 3.2.1.1 [ Aufgaben]Aufgaben<sup>3</sup>

---

<sup>3</sup>F, R, F, F, R



Richtig oder Falsch!?

1. `filter` filtert Spalten.
2. `filter` ist eine Funktion aus dem Paket `dplyr`.
3. `filter` erwartet als ersten Parameter das Filterkriterium.
4. `filter` lässt nur ein Filterkriterium zu.
5. Möchte man aus dem Datensatz `profiles` (`okcupiddata`) die Frauen filtern, so ist folgende Syntax korrekt: `filter(profiles, sex == "f")`.

### 3.2.1.2 Vertiefung: Fortgeschrittene Beispiele für `filter`

Einige fortgeschrittene Beispiele für `filter`:

Man kann alle Elemente (Zeilen) filtern, die zu einer Menge gehören und zwar mit diesem Operator: `%in%`:

```
filter(profiles, body_type %in% c("a little extra", "average"))
```

Besonders Textdaten laden zu einigen Extra-Überlegungen ein; sagen wir, wir wollen alle Personen filtern, die Katzen bei den Haustieren erwähnen. Es soll reichen, wenn `cat` ein Teil des Textes ist; also `likes dogs and likes cats` wäre OK (soll gefiltert werden). Dazu nutzen wir ein Paket zur Bearbeitung von Strings (Textdaten):

```
filter(profiles, str_detect(pets, "cats"))
```

Ein häufiger Fall ist, Zeilen *ohne* fehlende Werte (NAs) zu filtern. Das geht einfach:

```
profiles_keine_nas <- na.omit(profiles)
```

Aber was ist, wenn wir nur bei bestimmten Spalten wegen fehlender Werte besorgt sind? Sagen wir bei `income` und bei `sex`:

```
filter(profiles, !is.na(income) | !is.na(sex))
```

### 3.2.2 Spalten wählen mit `select`

Das Gegenstück zu `filter` ist `select`; dieser Befehl liefert die gewählten Spalten zurück. Das ist häufig praktisch, wenn der Datensatz sehr "breit" ist, also viele Spalten enthält. Dann kann es übersichtlicher sein, sich nur die relevanten auszuwählen. Abb. 5 zeigt Sinnbild für diesen Befehl:

vorher					nachher		
ID	Name	N1	N2	N3			
1	Anna	1	2	3	1	Anna	1
2	Berta	1	1	1	2	Berta	1
3	Carla	2	3	4	3	Carla	2
...	...	...	...	...	...	...	...

Abbildung 5: Spalten auswählen

Merke:

Die Funktion `select` wählt Spalten aus einem Dataframe aus.

Laden wir als ersten einen Datensatz.

```
stats_test <- read.csv("data/test_inf_short.csv")
```

Dieser Datensatz beinhaltet Daten zu einer Statistiklausur.

Beachten Sie, dass diese Syntax davon ausgeht, dass sich die Daten in einem Unterordner mit dem Namen `data` befinden, welcher sich im Arbeitsverzeichnis befindet<sup>4</sup>.

```
select(stats_test, score) # Spalte `score` auswählen
select(stats_test, score, study_time)
# Spalten `score` und `study_time` auswählen

select(stats_test, score:study_time) # dito
select(stats_test, 5:6) # Spalten 5 bis 6 auswählen
```

Tatsächlich ist der Befehl `select` sehr flexibel; es gibt viele Möglichkeiten, Spalten auszuwählen. Im `dplyr`-Cheatsheet findet sich ein guter Überblick dazu.

### 3.2.2.1 [ Aufgaben]Aufgaben<sup>5</sup>

<sup>4</sup>der angegebene Pfad ist also *relativ* zum aktuellen Verzeichnis.

<sup>5</sup>F, F, R, R, F



Richtig oder Falsch!?

1. `select` wählt *Zeilen* aus.
2. `select` ist eine Funktion aus dem Paket `knitr`.
3. Möchte man zwei Spalten auswählen, so ist folgende Syntax prinzipiell korrekt:  
`select(df, spalte1, spalte2)`.
4. Möchte man Spalten 1 bis 10 auswählen, so ist folgende Syntax prinzipiell korrekt:  
`'select(df, spalte1:spalte10)`
5. Mit `select` können Spalten nur bei ihrem Namen, aber nicht bei ihrer Nummer aufgerufen werden.

### 3.2.3 Zeilen sortieren mit `arrange`

Man kann zwei Arten des Umgangs mit R unterscheiden: Zum einen der “interaktive Gebrauch” und zum anderen “richtiges Programmieren”. Im interaktiven Gebrauch geht es uns darum, die Fragen zum aktuell vorliegenden Datensatz (schnell) zu beantworten. Es geht nicht darum, eine allgemeine Lösung zu entwickeln, die wir in die Welt verschicken können und die dort ein bestimmtes Problem löst, ohne dass der Entwickler (wir) dabei Hilfestellung geben muss. “Richtige” Software, wie ein R-Paket oder Microsoft Powerpoint, muss diese Erwartung erfüllen; “richtiges Programmieren” ist dazu vonnöten. Natürlich sind in diesem Fall die Ansprüche an die Syntax (der “Code”, hört sich cooler an) viel höher. In dem Fall muss man alle Eventualitäten voraussehen und sicherstellen, dass das Programm auch beim merkwürdigsten Nutzer brav seinen Dienst tut. Wir haben hier, beim interaktiven Gebrauch, niedrigere Ansprüche bzw. andere Ziele.

Beim interaktiven Gebrauch von R (oder beliebigen Analyseprogrammen) ist das Sortieren von Zeilen eine recht häufige Tätigkeit. Typisches Beispiel wäre der Lehrer, der eine Tabelle mit Noten hat und wissen will, welche Schüler die schlechtesten oder die besten sind in einem bestimmten Fach. Oder bei der Prüfung der Umsätze nach Filialen möchten wir die umsatzstärksten sowie -schwächsten Niederlassungen kennen.

Ein R-Befehl hierzu ist `arrange`; einige Beispiele zeigen die Funktionsweise am besten:

```
arrange(stats_test, score) # liefert die *schlechtesten* Noten zuerst zurück
arrange(stats_test, -score) # liefert die *besten* Noten zuerst zurück
arrange(stats_test, interest, score)
```

```
#>      X                V_1 study_time self_eval interest score
#> 1 234 23.01.2017 18:13:15         3         1         1      17
#> 2   4 06.01.2017 09:58:05         2         3         2      18
#> 3 131 19.01.2017 18:03:45         2         3         4      18
#> 4 142 19.01.2017 19:02:12         3         4         1      18
#> 5  35 12.01.2017 19:04:43         1         2         3      19
```

```
#> 6  71 15.01.2017 15:03:29      3      3      3    20
#>    X                      V_1 study_time self_eval interest score
#> 1   3 05.01.2017 23:33:47      5      10      6    40
#> 2   7 06.01.2017 14:25:49     NA      NA      NA    40
#> 3  29 12.01.2017 09:48:16      4      10      3    40
#> 4  41 13.01.2017 12:07:29      4      10      3    40
#> 5  58 14.01.2017 15:43:01      3       8      2    40
#> 6  83 16.01.2017 10:16:52     NA      NA      NA    40
#>    X                      V_1 study_time self_eval interest score
#> 1 234 23.01.2017 18:13:15      3       1      1     17
#> 2 142 19.01.2017 19:02:12      3       4      1     18
#> 3 221 23.01.2017 11:40:30      1       1      1     23
#> 4 230 23.01.2017 16:27:49      1       1      1     23
#> 5  92 17.01.2017 17:18:55      1       1      1     24
#> 6 107 18.01.2017 16:01:36      3       2      1     24
```

Einige Anmerkungen. Die generelle Syntax lautet `arrange(df, Spalte1, ...)`, wobei `df` den Dataframe bezeichnet und `Spalte1` die erste zu sortierende Spalte; die Punkte `...` geben an, dass man weitere Parameter übergeben kann. Man kann sowohl numerische Spalten als auch Textspalten sortieren. Am wichtigsten ist hier, dass man weitere Spalten übergeben kann. Dazu gleich mehr.

Standardmäßig sortiert `arrange` *aufsteigend* (weil kleine Zahlen im Zahlenstrahl vor den großen Zahlen kommen). Möchte man diese Reihenfolge umdrehen (große Werte zuert, d.h. *absteigend*), so kann man ein Minuszeichen vor den Namen der Spalte setzen.

Gibt man *zwei oder mehr* Spalten an, so werden pro Wert von `Spalte1` die Werte von `Spalte2` sortiert etc; man betrachte den Output des Beispiels oben dazu.

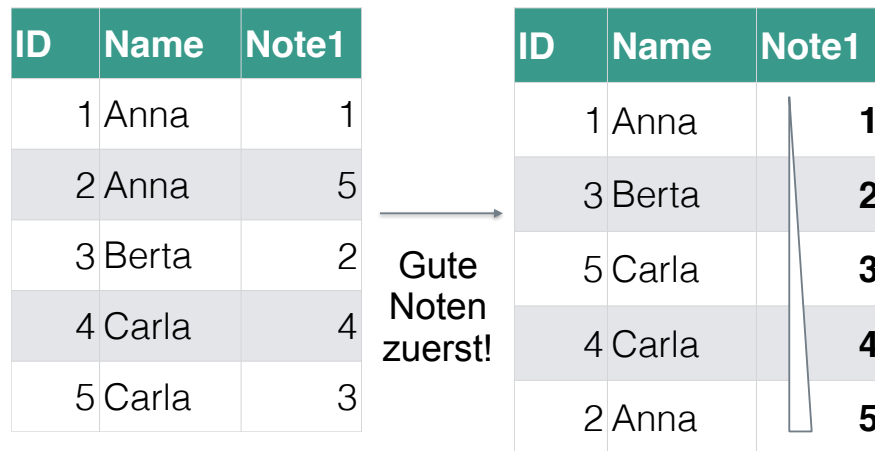
Merke:

Die Funktion `arrange` sortiert die Zeilen eines Dataframes.

Ein Sinnbild zur Verdeutlichung (s. Abb. 6):

Ein ähnliches Ergebnis erhält man mit `top_n()`, welches die *n größten Ränge* wiedergibt:

```
top_n(stats_test, 3)
#>    X                      V_1 study_time self_eval interest score
#> 1   3 05.01.2017 23:33:47      5      10      6    40
#> 2   7 06.01.2017 14:25:49     NA      NA      NA    40
#> 3  29 12.01.2017 09:48:16      4      10      3    40
#> 4  41 13.01.2017 12:07:29      4      10      3    40
#> 5  58 14.01.2017 15:43:01      3       8      2    40
#> 6  83 16.01.2017 10:16:52     NA      NA      NA    40
#> 7 116 18.01.2017 23:07:32      4       8      5    40
#> 8 119 19.01.2017 09:05:01     NA      NA      NA    40
```



ID	Name	Note1
1	Anna	1
2	Anna	5
3	Berta	2
4	Carla	4
5	Carla	3

Gute Noten zuerst!

ID	Name	Note1
1	Anna	<b>1</b>
3	Berta	<b>2</b>
5	Carla	<b>3</b>
4	Carla	<b>4</b>
2	Anna	<b>5</b>

Abbildung 6: Spalten sortieren

```
#> 9 132 19.01.2017 18:22:32 NA NA NA 40
#> 10 175 20.01.2017 23:03:36 5 10 5 40
#> 11 179 21.01.2017 07:40:05 5 9 1 40
#> 12 185 21.01.2017 15:01:26 4 10 5 40
#> 13 196 22.01.2017 13:38:56 4 10 5 40
#> 14 197 22.01.2017 14:55:17 4 10 5 40
#> 15 248 24.01.2017 16:29:45 2 10 2 40
#> 16 249 24.01.2017 17:19:54 NA NA NA 40
#> 17 257 25.01.2017 10:44:34 2 9 3 40
#> 18 306 27.01.2017 11:29:48 4 9 3 40
top_n(stats_test, 3, interest)
#> X V_1 study_time self_eval interest score
#> 1 3 05.01.2017 23:33:47 5 10 6 40
#> 2 5 06.01.2017 14:13:08 4 8 6 34
#> 3 43 13.01.2017 14:14:16 4 8 6 36
#> 4 65 15.01.2017 12:41:27 3 6 6 22
#> 5 110 18.01.2017 18:53:02 5 8 6 37
#> 6 136 19.01.2017 18:22:57 3 1 6 39
#> 7 172 20.01.2017 20:42:46 5 10 6 34
#> 8 214 22.01.2017 21:57:36 2 6 6 31
#> 9 301 27.01.2017 08:17:59 4 8 6 33
```

Gibt man *keine* Spalte an, so bezieht sich `top_n` auf die letzte Spalte im Datensatz.

Da sich hier mehrere Personen den größten Rang (Wert 40) teilen, bekommen wir *nicht* 3 Zeilen zurückgeliefert, sondern entsprechend mehr.



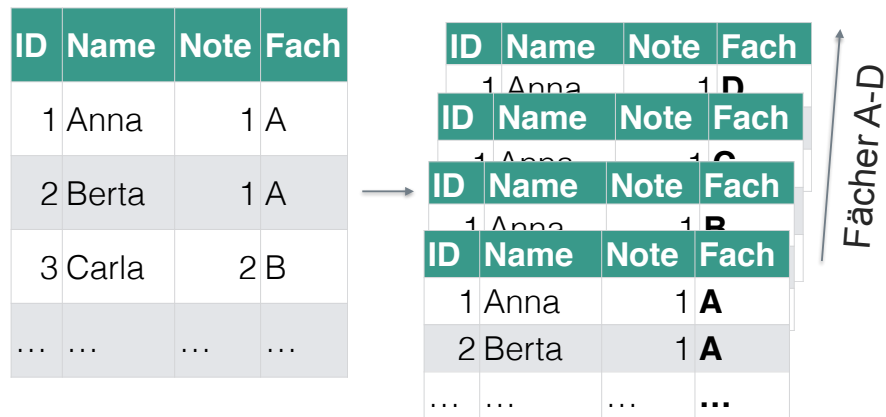


Abbildung 7: Datensätze nach Subgruppen aufteilen

### 3.2.3.1 [ Aufgaben]Aufgaben<sup>6</sup>



Richtig oder Falsch!?

1. `arrange` arrangiert Spalten.
2. `arrange` sortiert im Standard absteigend.
3. `arrange` lässt nur ein Sortierkriterium zu.
4. `arrange` kann numerische Werte, aber nicht Zeichenketten sortieren.
5. `top_n(5)` liefert die fünf kleinsten Ränge.

### 3.2.4 Datensatz gruppieren mit `group_by`

Einen Datensatz zu gruppieren ist eine häufige Angelegenheit: Was ist der mittlere Umsatz in Region X im Vergleich zu Region Y? Ist die Reaktionszeit in der Experimentalgruppe kleiner als in der Kontrollgruppe? Können Männer schneller ausparken als Frauen? Man sieht, dass das Gruppieren v.a. in Verbindung mit Mittelwerten oder anderen Zusammenfassungen sinnvoll ist; dazu im nächsten Abschnitt mehr.

Gruppieren meint, einen Datensatz anhand einer diskreten Variablen (z.B. Geschlecht) so aufzuteilen, dass Teil-Datensätze entstehen - pro Gruppe ein Teil-Datensatz (z.B. ein Datensatz, in dem nur Männer enthalten sind und einer, in dem nur Frauen enthalten sind).

In Abbildung 7 wurde der Datensatz anhand der Spalte (d.h. Variable) **Fach** in mehrere Gruppen geteilt (Fach A, Fach B...). Wir könnten uns als nächstes z.B. Mittelwerte pro Fach - d.h. pro Gruppe (pro Ausprägung von **Fach**) - ausgeben lassen; in diesem Fall vier Gruppen (Fach A bis D).

<sup>6</sup>F, F, F, F, R

```
test_gruppiert <- group_by(stats_test, interest)
test_gruppiert
#> Source: local data frame [306 x 6]
#> Groups: interest [7]
#>
#>      X                V_1 study_time self_eval interest score
#>   <int>             <fctr>    <int>    <int>    <int> <int>
#> 1     1 05.01.2017 13:57:01         5         8         5    29
#> 2     2 05.01.2017 21:07:56         3         7         3    29
#> 3     3 05.01.2017 23:33:47         5        10         6    40
#> 4     4 06.01.2017 09:58:05         2         3         2    18
#> 5     5 06.01.2017 14:13:08         4         8         6    34
#> 6     6 06.01.2017 14:21:18        NA        NA        NA    39
#> 7     7 06.01.2017 14:25:49        NA        NA        NA    40
#> 8     8 06.01.2017 17:24:53         2         5         3    24
#> 9     9 07.01.2017 10:11:17         2         3         5    25
#> 10    10 07.01.2017 18:10:05         4         5         5    33
#> # ... with 296 more rows
```

Schaut man sich nun den Datensatz an, sieht man erstmal wenig Effekt der Gruppierung. R teilt uns lediglich mit `Groups: interest [7]`, dass es 7 Gruppen gibt, aber es gibt keine extra Spalte oder sonstige Anzeichen der Gruppierung. Aber keine Sorge, wenn wir gleich einen Mittelwert ausrechnen, bekommen wir den Mittelwert pro Gruppe!

Ein paar Hinweise: `Source: local data frame [306 x 6]` will sagen, dass die Ausgabe sich auf einen `tibble` bezieht<sup>7</sup>, also eine bestimmte Art von Dataframe. `Groups: interest [7]` zeigt, dass der Tibble in 7 Gruppen - entsprechend der Werte von `interest` aufgeteilt ist.

`group_by` an sich ist nicht wirklich nützlich. Nützlich wird es erst, wenn man weitere Funktionen auf den gruppierten Datensatz anwendet - z.B. Mittelwerte ausrechnet (z.B. mit `summarise`, s. unten). Die nachfolgenden Funktionen (wenn sie aus `dplyr` kommen), berücksichtigen nämlich die Gruppierung. So kann man einfach Mittelwerte pro Gruppe ausrechnen. `dplyr` kombiniert dann die Zusammenfassungen (z.B. Mittelwerte) der einzelnen Gruppen in einen Dataframe und gibt diesen dann aus.

Die Idee des “Gruppieren - Zusammenfassen - Kombinieren” ist flexibel; man kann sie häufig brauchen. Es lohnt sich, diese Idee zu lernen (vgl. Abb. 8).

### 3.2.4.1 [Aufgaben]Aufgaben<sup>8</sup>

<sup>7</sup><http://stackoverflow.com/questions/29084380/what-is-the-meaning-of-the-local-data-frame-message-fro>

<sup>8</sup>R, F, R, R

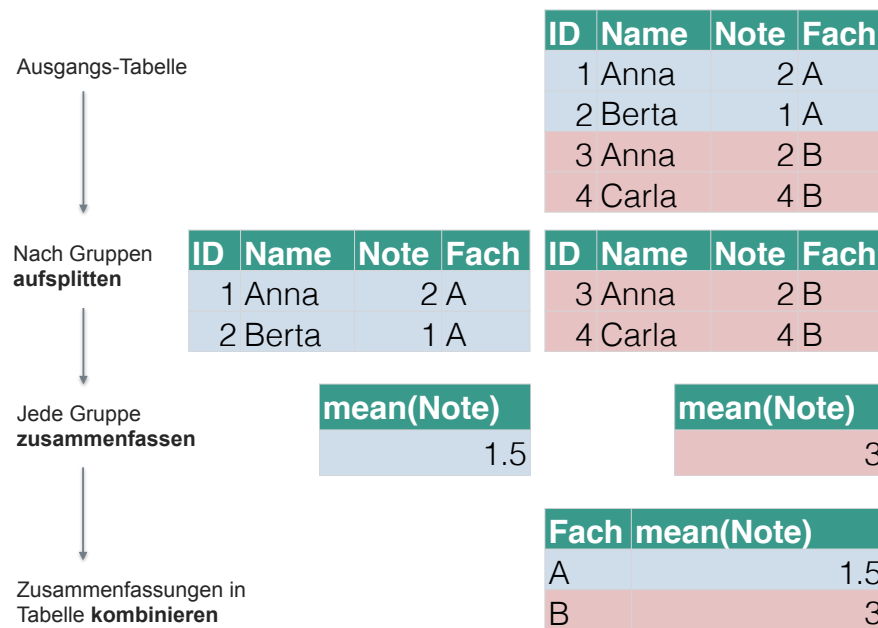


Abbildung 8: Schematische Darstellung des 'Gruppieren - Zusammenfassen - Kombinieren'



Richtig oder Falsch!?

1. Mit `group_by` gruppiert man einen Datensatz.
2. `group_by` lässt nur ein Gruppierungskriterium zu.
3. Die Gruppierung durch `group_by` wird nur von Funktionen aus `dplyr` erkannt.
4. `group_by` ist sinnvoll mit `summarise` zu kombinieren.

Merke:

Mit `group_by` teilt man einen Datensatz in Gruppen ein, entsprechend der Werte einer mehrerer Spalten.

### 3.2.5 Eine Spalte zusammenfassen mit `summarise`

Vielleicht die wichtigste oder häufigste Tätigkeit in der Analyse von Daten ist es, eine Spalte zu *einem* Wert zusammenzufassen; `summarise` leistet dies. Anders gesagt: Einen Mittelwert berechnen, den größten (kleinsten) Wert herausuchen, die Korrelation berechnen oder eine beliebige andere Statistik ausgeben lassen. Die Gemeinsamkeit dieser Operationen ist, dass sie eine Spalte zu einem Wert zusammenfassen, "aus Spalte mach Zahl", sozusagen. Daher ist der Name des Befehls `summarise` ganz passend. Genauer gesagt fasst dieser Befehl eine Spalte zu einer Zahl zusammen *anhand* einer Funktion wie `mean` oder `max` (vgl. Abb. 9. Hierbei ist jede Funktion erlaubt, die eine Spalte als Input verlangt und eine Zahl zurückgibt; andere Funktionen sind bei `summarise` nicht erlaubt.

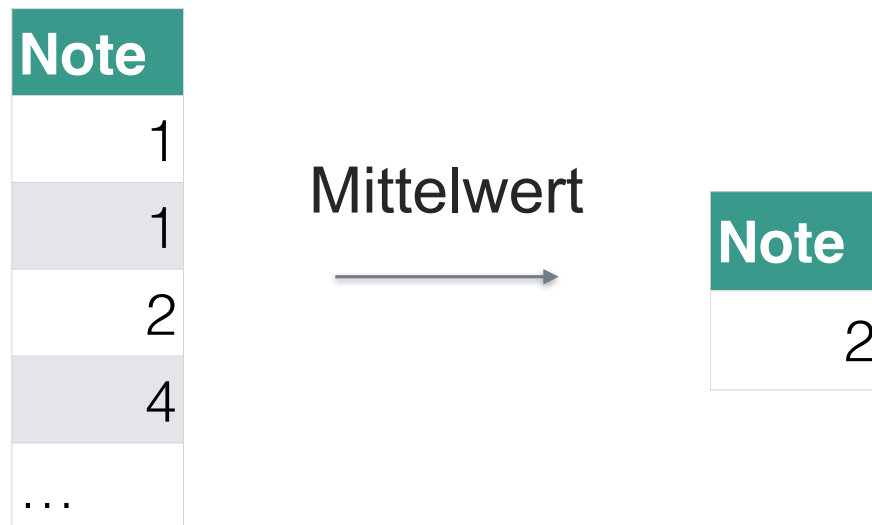


Abbildung 9: Spalten zu einer Zahl zusammenfassen

```
summarise(stats_test, mean(score))
#>   mean(score)
#> 1         31.1
```

Man könnte diesen Befehl so ins Deutsche übersetzen: Fasse aus Tabelle `stats_test` die Spalte `score` anhand des Mittelwerts zusammen. Nicht vergessen, wenn die Spalte `score` fehlende Werte hat, wird der Befehl `mean` standardmäßig dies mit `NA` quittieren. Ergänzt man den Parameter `na.rm = TRUE`, so ignoriert R fehlende Werte und der Befehl `mean` liefert ein Ergebnis zurück.

Jetzt können wir auch die Gruppierung nutzen:

```
test_gruppiert <- group_by(stats_test, interest)
summarise(test_gruppiert, mean(score, na.rm = TRUE))
#> # A tibble: 7 × 2
#>   interest `mean(score, na.rm = TRUE)`
#>   <int>      <dbl>
#> 1     1      28.3
#> 2     2      29.7
#> 3     3      30.8
#> 4     4      29.9
#> 5     5      32.5
#> 6     6      34.0
#> 7     NA      33.1
```

Der Befehl `summarise` erkennt also, wenn eine (mit `group_by`) gruppierte Tabelle vorliegt. Jegliche Zusammenfassung, die wir anfordern, wird anhand der Gruppierungsinformation

aufgeteilt werden. In dem Beispiel bekommen wir einen Mittelwert für jeden Wert von `interest`. Interessanterweise sehen wir, dass der Mittelwert tendenziell größer wird, je größer `interest` wird.

Alle diese `dplyr`-Befehle geben einen Dataframe zurück, was praktisch ist für weitere Verarbeitung. In diesem Fall heißen die Spalten `interest` und `mean(score)`. Zweiter Name ist nicht so schön, daher ändern wir den wie folgt:

Jetzt können wir auch die Gruppierung nutzen:

```
test_gruppiert <- group_by(stats_test, interest)
summarise(test_gruppiert, mw_pro_gruppe = mean(score, na.rm = TRUE))
#> # A tibble: 7 × 2
#>   interest mw_pro_gruppe
#>   <int>     <dbl>
#> 1     1         28.3
#> 2     2         29.7
#> 3     3         30.8
#> 4     4         29.9
#> 5     5         32.5
#> 6     6         34.0
#> 7    NA         33.1
```

Nun heißt die zweite Spalte `mw_pro_Gruppe`. `na.rm = TRUE` veranlasst, bei fehlenden Werten trotzdem einen Mittelwert zurückzuliefern (die Zeilen mit fehlenden Werten werden in dem Fall ignoriert).

Grundsätzlich ist die Philosophie der `dplyr`-Befehle: “Mach nur eine Sache, aber die dafür gut”. Entsprechend kann `summarise` nur *Spalten* zusammenfassen, aber keine *Zeilen*.

Merke:

Mit `summarise` kann man eine Spalte eines Dataframes zu einem Wert zusammenfassen.

### 3.2.5.1 Deskriptive Statistik mit `summarise`

Die deskriptive Statistik hat zwei Haupt-Bereiche: Lagemaße und Streuungsmaße.

*Lagemaße* geben den “typischen”, “mittleren” oder “repräsentativen” Vertreter der Verteilung an. Bei den Lagemaßen denkt man sofort an das *arithmetische Mittel* (synonym: Mittelwert; häufig als  $\bar{X}$  abgekürzt; `mean`). Ein Nachteil von Mittelwerten ist, dass sie nicht robust gegenüber Extremwerte sind: Schon ein vergleichsweise großer Einzelwert kann den Mittelwert deutlich verändern und damit die Repräsentativität des Mittelwerts für die Gesamtmenge der Daten in Frage stellen. Eine robuste Variante ist der *Median* (Md; `median`). Ist die Anzahl der (unterschiedlichen) Ausprägungen nicht zu groß im Verhältnis zur Fallzahl, so ist der

*Modus* eine sinnvolle Statistik; er gibt die häufigste Ausprägung an<sup>9</sup>.

*Streuungsmaße* geben die Unterschiedlichkeit in den Daten wieder; mit anderen Worten: sind die Daten sich ähnlich oder unterscheiden sich die Werte deutlich? Zentrale Statistiken sind der *mittlere Absolutabstand* (MAA; MAD),<sup>10</sup> die *Standardabweichung* (sd; `sd`), die *Varianz* (Var; `var`) und der *Interquartilsabstand* (IQR; `IQR`). Da nur der IQR *nicht* auf dem Mittelwert basiert, ist er am robustesten. Beliebige Quantile bekommt man mit dem R-Befehl `quantile`.

Der Befehl `summarise` eignet sich, um deskriptive Statistiken auszurechnen.

```
summarise(stats_test, mean(score))
#>   mean(score)
#> 1         31.1
summarise(stats_test, sd(score))
#>   sd(score)
#> 1         5.74
```

Natürlich könnte man auch einfacher schreiben:

```
mean(stats_test$score)
#> [1] 31.1
median(stats_test$score)
#> [1] 31
```

`summarise` liefert aber im Unterschied zu `mean` etc. immer einen Dataframe zurück. Da der Dataframe die typische Datenstruktur ist, ist es häufig praktisch, wenn man einen Dataframe zurückbekommt, mit dem man weiterarbeiten kann. Außerdem lassen `mean` etc. keine Gruppierungsoperationen zu; über `group_by` kann man dies aber bei `dplyr` erreichen.

### 3.2.5.2 [ Aufgaben]Aufgaben<sup>11</sup>



Richtig oder Falsch!?

1. Möchte man aus der Tabelle `stats_test` den Mittelwert für die Spalte `score` berechnen, so ist folgende Syntax korrekt: `summarise(stats_test, mean(score))`.
2. `summarise` liefert eine Tabelle, genauer: einen Tibble, zurück.
3. Die Tabelle, die diese Funktion zurückliefert: `summarise(stats_test, mean(score))`, hat eine Spalte mit dem Namen `mean(score)`.

<sup>9</sup>Der *Modus* ist im Standard-R nicht mit einem eigenen Befehl vertreten. Man kann ihn aber leicht von Hand bestimmen; s.u. Es gibt auch einige Pakete, die diese Funktion anbieten: z.B. <https://cran.r-project.org/web/packages/modes/index.html>

<sup>10</sup>Der *MAD* ist im Standard-R nicht mit einem eigenen Befehl vertreten. Es gibt einige Pakete, die diese Funktion anbieten: z.B. <https://artax.karlin.mff.cuni.cz/r-help/library/lsr/html/aad.html>

<sup>11</sup>R, R, R, R, R

4. `summarise` lässt zu, dass die zu berechnende Spalte einen Namen vom Nutzer zugewiesen bekommt.
5. `summarise` darf nur verwendet werden, wenn eine Spalte zu einem Wert zusammengefasst werden soll.

1. (Fortgeschritten) Bauen Sie einen eigenen Weg, um den mittleren Absolutabstand auszurechnen! Gehen Sie der Einfachheit halber (zuerst) von einem Vektor mit den Werten (1,2,3) aus!

Lösung:

```
x <- c(1, 2, 3)
x_mw <- mean(x)
x_delta <- x - x_mw
x_delta <- abs(x_delta)
mad <- mean(x_delta)
mad
#> [1] 0.667
```

### 3.2.6 Zeilen zählen mit `n` und `count`

Ebenfalls nützlich ist es, Zeilen zu zählen. Im Gegensatz zum Standardbefehl<sup>12</sup> `nrow` versteht der `dplyr`-Befehl `n` auch Gruppierungen. `n` darf nur innerhalb von `summarise` oder ähnlichen `dplyr`-Befehlen verwendet werden.

```
summarise(stats_test, n())
#>   n()
#> 1 306
summarise(test_gruppiert, n())
#> # A tibble: 7 × 2
#>   interest `n()`
#>   <int> <int>
#> 1     1    30
#> 2     2    47
#> 3     3    66
#> 4     4    41
#> 5     5    45
#> 6     6     9
#> 7    NA    68
```

<sup>12</sup>Standardbefehl meint, dass die Funktion zum Standardrepertoire von R gehört, also nicht über ein Paket extra geladen werden muss

```
nrow(stats_test)
#> [1] 306
```

Außerhalb von gruppierten Datensätzen ist `nrow` meist praktischer.

Praktischer ist der Befehl `count`, der nichts anderes ist als die Hintereinanderschaltung von `group_by` und `n`. Mit `count` zählen wir die Häufigkeiten nach Gruppen; Gruppen sind hier zumeist die Werte einer auszuzählenden Variablen (oder mehrerer auszuzählender Variablen). Das macht `count` zu einem wichtigen Helfer bei der Analyse von Häufigkeitsdaten.

```
dplyr::count(stats_test, interest)
#> # A tibble: 7 × 2
#>   interest     n
#>   <int> <int>
#> 1       1    30
#> 2       2    47
#> 3       3    66
#> 4       4    41
#> 5       5    45
#> 6       6     9
#> 7      NA    68

dplyr::count(stats_test, study_time)
#> # A tibble: 6 × 2
#>   study_time     n
#>   <int> <int>
#> 1       1    31
#> 2       2    49
#> 3       3    85
#> 4       4    56
#> 5       5    17
#> 6      NA    68

dplyr::count(stats_test, interest, study_time)
#> # A tibble: 29 × 3
#>   interest study_time     n
#>   <int>      <int> <int>
#> 1       1         1    12
#> 2       1         2     7
#> 3       1         3     8
#> 4       1         4     2
#> 5       1         5     1
#> 6       2         1     9
#> 7       2         2    15
#> 8       2         3    16
```



```
#> 9      2      4      6
#> 10     2      5      1
#> # ... with 19 more rows
```

Allgemeiner formuliert lautet die Syntax: `count(df, Spalte1, ...)`, wobei `df` der Dataframe ist und `Spalte1` die erste (es können mehrere sein) auszuzählende Spalte. Gibt man z.B. zwei Spalten an, so wird pro Wert der 1. Spalte die Häufigkeiten der 2. Spalte ausgegeben.

Merke:

`n` und `count` zählen die Anzahl der Zeilen, d.h. die Anzahl der Fälle.

### 3.2.6.1 [Aufgaben]Aufgaben<sup>13</sup>



Richtig oder Falsch!?

1. Mit `count` kann man Zeilen zählen.
2. `count` ist ähnlich (oder identisch) zu einer Kombination von `group_by` und `n()`.
3. Mit `count` kann man nur eine Gruppe beim Zählen berücksichtigen.
4. `count` darf nicht bei nominalskalierten Variablen verwendet werden.

1. Bauen Sie sich einen Weg, um den Modus mithilfe von `count` und `arrange` zu bekommen!

```
stats_count <- count(stats_test, score)
stats_count_sortiert <- arrange(stats_count, -n)
head(stats_count_sortiert, 1)
#> # A tibble: 1 × 2
#>   score     n
#>   <int> <int>
#> 1     34    22
```

Ah! Der Score 34 ist der häufigste!

## 3.3 Die Pfeife

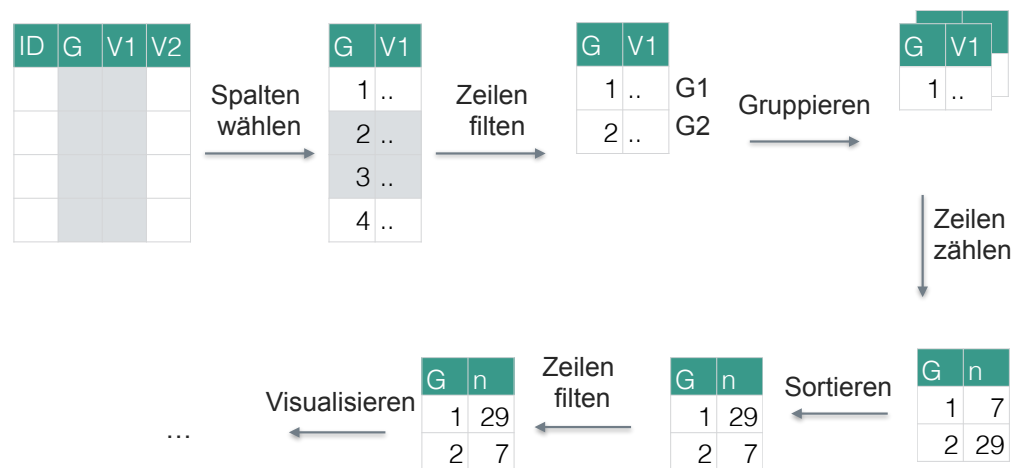
Die zweite Idee kann man salopp als “Durchpfeifen” oder die “Idee der Pfeife bezeichnen; ikonographisch mit einem Pfeifen ähnlichen Symbol dargestellt `%>%`. Der Begriff “Durchpfeifen” ist frei vom Englischen “to pipe” übernommen. Das berühmte Bild von René Magritte stand dabei Pate (s. Abb. 10).

Hierbei ist gemeint, einen Datensatz sozusagen auf ein Fließband zu legen und an jedem Arbeitsplatz einen Arbeitsschritt auszuführen. Der springende Punkt ist, dass ein Dataframe

<sup>13</sup>R, R, F, F



**Abbildung 10:** La trahison des images [Ceci n'est pas une pipe], René Magritte, 1929, © C. Herscovici, Brussels / Artists Rights Society (ARS), New York, <http://collections.lacma.org/node/239578>



**Abbildung 11:** Das 'Durchpeifen'

als "Rohstoff" eingegeben wird und jeder Arbeitsschritt seinerseits wieder einen Dataframe ausgiebt. Damit kann man sehr schön, einen "Flow" an Verarbeitung erreichen, außerdem spart man sich Tipparbeit und die Syntax wird lesbarer. Damit das Durchpeifen funktioniert, benötigt man Befehle, die als Eingabe einen Dataframe erwarten und wieder einen Dataframe zurückliefern. Das Schaubild verdeutlicht beispielhaft eine Abfolge des Durchpeifens (s. Abb. 11).

Die sog. "Pfeife" (pipe: `%>%`) in Anspielung an das berühmte Bild von René Magritte, verkettet Befehle hintereinander. Das ist praktisch, da es die Syntax vereinfacht. Vergleichen Sie mal diese Syntax

```
filter(summarise(group_by(filter(stats_test, !is.na(score)), interest), mw = mean(score
```

mit dieser

```
stats_test %>%
  filter(!is.na(score)) %>%
```

```

group_by(interest) %>%
  summarise(mw = mean(score)) %>%
  filter(mw > 30)
#> # A tibble: 4 × 2
#>   interest    mw
#>   <int> <dbl>
#> 1     3  30.8
#> 2     5  32.5
#> 3     6  34.0
#> 4    NA  33.1

```

Es ist hilfreich, diese “Pfeifen-Syntax” in deutschen Pseudo-Code zu übersetzen.



Nimm die Tabelle “stats\_test” UND DANN  
 filtere alle nicht-fehlenden Werte UND DANN  
 gruppier die verbleibenden Werte nach “interest” UND DANN  
 bilde den Mittelwert (pro Gruppe) für “score” UND DANN  
 liefere nur die Werte größer als 30 zurück.

Die zweite Syntax, in “Pfeifenform” ist viel einfacher zu verstehen als die erste! Die erste Syntax ist verschachtelt, man muss sie von innen nach außen lesen. Das ist kompliziert. Die Pfeife in der 2. Syntax macht es viel einfacher, die Syntax zu verstehen, da die Befehle “hintereinander” gestellt (sequenziell organisiert) sind.

Die Pfeife zerlegt die “russische Puppe”, also ineinander verschachtelten Code, in sequenzielle Schritte und zwar in der richtigen Reihenfolge (entsprechend der Abarbeitung). Wir müssen den Code nicht mehr von innen nach außen lesen (wie das bei einer mathematischen Formel der Fall ist), sondern können wie bei einem Kochrezept “erstens ..., zweitens .., drittens ...” lesen. Die Pfeife macht die Syntax einfacher. Natürlich hätten wir die verschachtelte Syntax in viele einzelne Befehle zerlegen können und jeweils eine Zwischenergebnis speichern mit dem Zuweisungspfeil <- und das Zwischenergebnis dann explizit an den nächsten Befehl weitergeben. Eigentlich macht die Pfeife genau das - nur mit weniger Tipparbeit. Und auch einfacher zu lesen. Flow!



Wenn Sie Befehle verketteten mit der Pfeife, sind nur Befehle erlaubt, die einen Datensatz als Eingabe verlangen und einen Datensatz ausgeben. Das ist bei den hier vorgestellten Funktionen der Fall. Viele andere Funktionen erfüllen dieses Kriterium aber nicht; in dem Fall liefert `dplyr` eine Fehlermeldung.

### 3.3.1 Spalten berechnen mit `mutate`

Wenn man die Pfeife benutzt, ist der Befehl `mutate` ganz praktisch: Er berechnet eine Spalte. Normalerweise kann man einfach eine Spalte berechnen mit dem Zuweisungsoperator:

Zum Beispiel so:

```
df$neue_spalte <- df$spalte1 + df$spalte2
```

Innerhalb einer Pfeifen-Syntax geht das aber nicht (so gut). Da ist man mit der Funktion `mutate` besser beraten; `mutate` leistet just dasselbe wie die Pseudo-Syntax oben:

```
df %>%
  mutate(neue_spalte = spalte1 + spalte2)
```

In Worten:



Nimm die Tabelle “df” UND DANN bilde eine neue Spalte mit dem Namen `neue_spalte`, die sich berechnet als Summe von `spalte1` und `spalte2`.

Allerdings berücksichtigt `mutate` auch Gruppierungen. Der Hauptvorteil ist die bessere Lesbarkeit durch Auflösen der Verschachtelungen.

Ein konkretes Beispiel:

```
stats_test %>%
  mutate(bestanden = score > 25) %>%
  head()
```

#>	X	V_1	study_time	self_eval	interest	score	bestanden	
#> 1	1	05.01.2017	13:57:01	5	8	5	29	TRUE
#> 2	2	05.01.2017	21:07:56	3	7	3	29	TRUE
#> 3	3	05.01.2017	23:33:47	5	10	6	40	TRUE
#> 4	4	06.01.2017	09:58:05	2	3	2	18	FALSE
#> 5	5	06.01.2017	14:13:08	4	8	6	34	TRUE
#> 6	6	06.01.2017	14:21:18	NA	NA	NA	39	TRUE

Diese Syntax erzeugt eine neue Spalte innerhalb von `stats_test`; diese Spalte prüft pro Person, ob `score > 25` ist. Falls ja (TRUE), dann ist `bestanden` TRUE, ansonsten ist `bestanden` FALSE (Pech). `head` zeigt die ersten 6 Zeilen des resultierenden Dataframes an.

Abb. 12 zeigt Sinnbild für `mutate`:

### 3.3.2 Aufgaben

1. Entschlüsseln Sie dieses Ungetüm! Übersetzen Sie diese Syntax auf Deutsch:

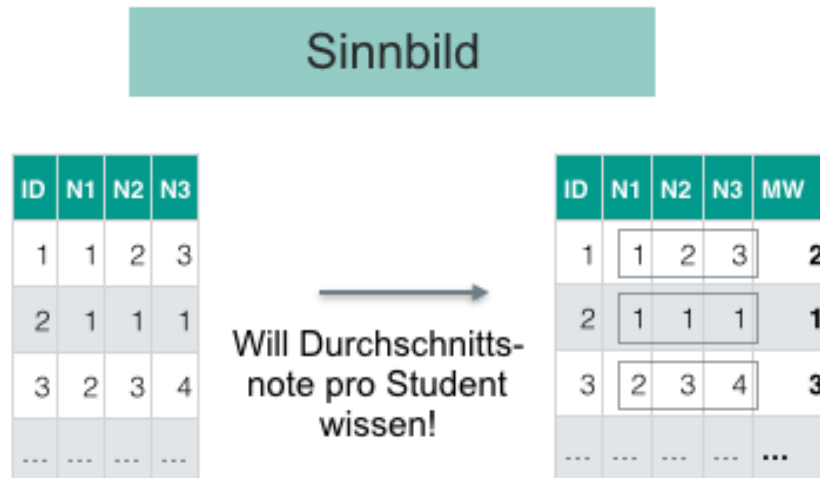


Abbildung 12: Sinnbild für mutate

```
library(nycflights13)
data(flights)

verspaetung <-
  filter(
    summarise(
      group_by(filter(flights, !is.na(dep_delay), month)), delay = mean(dep_delay), n = n
```

2. Entschlüsseln Sie jetzt diese Syntax bzw. übersetzen Sie sie ins Deutsche:

```
verspaetung <- flights %>% filter(!is.na(dep_delay)) %>%
  group_by(month) %>%
  summarise(delay = mean(dep_delay), n = n()) %>% filter(n > 10)
```

3. (schwierig) Die Pfeife bei `arr_delay`

- Übersetzen Sie die folgende Pseudo-Syntax ins ERRRische!



Nehme den Datensatz `flights` UND DANN...  
 Wähle daraus die Spalte `arr_delay` UND DANN...  
 Berechne den Mittelwert der Spalte UND DANN...  
 ziehe vom Mittelwert die Spalte ab UND DANN... quadriere die einzelnen Differenzen  
 UND DANN... bilde davon den Mittelwert.

Lösung:

```

flights %>%
  select(arr_delay) %>%
  mutate(arr_delay_delta = arr_delay - mean(flights$arr_delay, na.rm = TRUE)) %>%
  mutate(arr_delay_delta_quadrat = arr_delay_delta^2) %>%
  summarise(arr_delay_var = mean(arr_delay_delta_quadrat, na.rm = TRUE)) %>%
  summarise(sqrt(arr_delay_var))
#> # A tibble: 1 × 1
#>   `sqrt(arr_delay_var)`
#>               <dbl>
#> 1                44.6

```

- Berechnen Sie die sd von `arr_delay` in `flights`! Vergleichen Sie sie mit dem Ergebnis der vorherigen Aufgabe!<sup>14</sup>
- Was hat die Pfeifen-Syntax oben berechnet?<sup>15</sup>

### 3.4 Befehlsübersicht

Paket::Funktion	Beschreibung
<code>dplyr::arrange</code>	Sortiert Spalten
<code>dplyr::filter</code>	Filtiert Zeilen
<code>dplyr::select</code>	Wählt Spalten
<code>dplyr::group_by</code>	gruppiert einen Dataframe
<code>dplyr::n</code>	zählt Zeilen
<code>dplyr::count</code>	zählt Zeilen nach Untergruppen
<code>%&gt;% (dplyr)</code>	verkettet Befehle
<code>dplyr::mutate</code>	erzeugt/berechnet Spalten

### 3.5 Verweise

- Die offizielle Dokumentation von `dplyr` findet sich hier: <https://cran.r-project.org/web/packages/dplyr/dplyr.pdf>.
- Eine schöne Demonstration der Mächtigkeit von `dplyr` findet sich hier: <http://bit.ly/2kX9lvC>.
- Die GUI “exploratory” ist ein “klickbare” Umsetzung von `dplyr`, mächtig, modern und sieht cool aus: <https://exploratory.io>.
- *R for Data Science* bietet umfangreiche Unterstützung zu diesem Thema (Wickham und Grolemund 2016).

<sup>14</sup>`sd(flights$arr_delay, na.rm = TRUE)`

<sup>15</sup>die sd von `arr_delay`

- 4 Praxisprobleme der Datenaufbereitung
- 5 Fallstudie zum Datenjudo
- 6 Daten visualisieren
- 7 Fallstudie zur Visualisierung
- 8 Grundlagen des Modellierens
- 9 Der p-Wert
- 10 Klassische lineare (numerische) Regression
- 11 Klassifizierende Regression
- 12 Fallstudien zum geleiteten Modellieren
- 13 nicht robust:
- 14 Vertiefung: Clusteranalyse
- 15 Probeklausur
- 16 Literaturverzeichnis

Allaire, JJ, Joe Cheng, Yihui Xie, Jonathan McPherson, Winston Chang, Jeff Allen, Hadley Wickham, Aron Atkins, und Rob Hyndman. 2016a. *rmarkdown: Dynamic Documents for R*. <https://CRAN.R-project.org/package=rmarkdown>.

———. 2016b. *rmarkdown: Dynamic Documents for R*. <https://CRAN.R-project.org/package=rmarkdown>.

Auguie, Baptiste. 2016. *gridExtra: Miscellaneous Functions for „Grid“ Graphics*. <https://CRAN.R-project.org/package=gridExtra>.

[//CRAN.R-project.org/package=gridExtra](https://CRAN.R-project.org/package=gridExtra).

Beaujean, A. Alexander. 2012. *BaylorEdPsych: R Package for Baylor University Educational Psychology Quantitative Courses*. <https://CRAN.R-project.org/package=BaylorEdPsych>.

Benoit, Kenneth, und Paul Nulty. 2016. *quanteda: Quantitative Analysis of Textual Data*. <https://CRAN.R-project.org/package=quanteda>.

Bouchet-Valat, Milan. 2014. *SnowballC: Snowball stemmers based on the C libstemmer UTF-8 library*. <https://CRAN.R-project.org/package=SnowballC>.

Briggs, William M. 2008. *Breaking the Law of Averages: Real-Life Probability and Statistics in Plain English*. Lulu.com. <https://www.amazon.com/Breaking-Law-Averages-Probability-Statistics/dp/0557019907%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0557019907>.

———. 2016. *Uncertainty: The Soul of Modeling, Probability & Statistics*. Springer. <https://www.amazon.com/Uncertainty-Soul-Modeling-Probability-Statistics-ebook/dp/B01JEJNUJK%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3DB01JEJNUJK>.

Bryant, PG, und MA Smith. 1995. „Practical Data Analysis: Case Studies in Business Statistics, Homewood, IL: Richard D“. Irwin Publishing.

Chang, Winston. 2015. *downloader: Download Files over HTTP and HTTPS*. <https://CRAN.R-project.org/package=downloader>.

Chapman, Chris, und Elea McDonnell Feit. 2015. *R for Marketing Research and Analytics*. Springer International Publishing. doi:10.1007/978-3-319-14436-8<sup>16</sup>.

Cleveland, William S. 1993. *Visualizing Data*. Hobart Press. <https://www.amazon.com/Visualizing-Data-William-S-Cleveland/dp/0963488406%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0963488406>.

Cobb, George W. 2007. „The introductory statistics course: a Ptolemaic curriculum?“ *Technology Innovations in Statistics Education* 1 (1).

Cortez, Paulo, António Cerdeira, Fernando Almeida, Telmo Matos, und José Reis. 2009. „Modeling wine preferences by data mining from physicochemical properties“. *Decision Support Systems* 47 (4). Elsevier: 547–53.

de Vries, Andrie, und Brian D. Ripley. 2016. *ggdendro: Create Dendrograms and Tree Diagrams Using 'ggplot2'*. <https://CRAN.R-project.org/package=ggdendro>.

Feinerer, Ingo, und Kurt Hornik. 2015. *tm: Text Mining Package*. <https://CRAN.R-project.org/package=tm>.

Fellows, Ian. 2014. *wordcloud: Word Clouds*. <https://CRAN.R-project.org/package=>

---

<sup>16</sup><https://doi.org/10.1007/978-3-319-14436-8>



wordcloud.

Fox, John, und Sanford Weisberg. 2016. *car: Companion to Applied Regression*. <https://CRAN.R-project.org/package=car>.

Gigerenzer, Gerd. 1980. *Messung und Modellbildung in der Psychologie (Uni-Taschenbücher. Psychologie, Pädagogik, Soziologie, Psychiatrie) (German Edition)*. E. Reinhardt. <https://www.amazon.com/Modellbildung-Psychologie-Uni-Taschenbuecher-Soziologie-Psychiatrie/dp/3497008958%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dteckie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D3497008958>.

———. 2004. „Mindless statistics“. *The Journal of Socio-Economics* 33 (5). Elsevier BV: 587–606. doi:10.1016/j.socec.2004.09.033<sup>17</sup>.

Grolemund, Garrett, und Hadley Wickham. 2014. „A cognitive interpretation of data analysis“. *International Statistical Review* 82 (2). Wiley Online Library: 184–204.

Hahsler, Michael, Christian Buchta, Bettina Gruen, und Kurt Hornik. 2016. *arules: Mining Association Rules and Frequent Itemsets*. <https://CRAN.R-project.org/package=arules>.

Hahsler, Michael, und Sudheer Chelluboina. 2016. *arulesViz: Visualizing Association Rules and Frequent Itemsets*. <https://CRAN.R-project.org/package=arulesViz>.

Hamermesh, Daniel S, und Amy Parker. 2005. „Beauty in the classroom: Instructors’ pulchritude and putative pedagogical productivity“. *Economics of Education Review* 24 (4). Elsevier: 369–76.

Hardin, Johanna, Roger Hoerl, Nicholas J Horton, Deborah Nolan, Ben Baumer, Olaf Hall-Holt, Paul Murrell, u. a. 2015. „Data science in statistics curricula: Preparing students to ‚Think with Data‘“. *The American Statistician* 69 (4). Taylor & Francis: 343–53.

Head, Megan L., Luke Holman, Rob Lanfear, Andrew T. Kahn, und Michael D. Jennions. 2015. „The Extent and Consequences of P-Hacking in Science“. *PLOS Biology* 13 (3). Public Library of Science (PLOS): e1002106. doi:10.1371/journal.pbio.1002106<sup>18</sup>.

Hendricks, Paul. 2015. *titanic: Titanic Passenger Survival Data Set*. <https://CRAN.R-project.org/package=titanic>.

Ingo Feinerer, Kurt Hornik, und David Meyer. 2008. „Text Mining Infrastructure in R“. *Journal of Statistical Software* 25 (5): 1–54. <http://www.jstatsoft.org/v25/i05/>.

Jackson, Simon. 2016. *corrr: Correlations in R*. <https://CRAN.R-project.org/package=corrr>.

James, Gareth, Daniela Witten, Trevor Hastie, und Rob Tibshirani. 2013a. *ISLR: Data for An Introduction to Statistical Learning with Applications in R*. <https://CRAN.R-project.org/package=ISLR>.

James, Gareth, Daniela Witten, Trevor Hastie, und Robert Tibshirani. 2013b. *An introduction*

<sup>17</sup><https://doi.org/10.1016/j.socec.2004.09.033>

<sup>18</sup><https://doi.org/10.1371/journal.pbio.1002106>

*to statistical learning*. Bd. 6. Springer.

———. 2013c. *An introduction to statistical learning*. Bd. 6. Springer.

Julia, PhD Silge, und PhD Robinson David. 2017. *Text Mining with R: A tidy approach*. O'Reilly Media. <https://www.amazon.com/Text-Mining-R-tidy-approach/dp/1491981652%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1491981652>.

Kim, Albert Y, und Adriana Escobedo-Land. 2015. „OkCupid Data for Introductory Statistics and Data Science Courses“. *Journal of Statistics Education* 23 (2). Citeseer: n2.

Kim, Albert Y., und Adriana Escobedo-Land. 2016. *okcupiddata: OkCupid Profile Data for Introductory Statistics and Data Science Courses*. <https://CRAN.R-project.org/package=okcupiddata>.

Krämer, W. 2011. *Wie wir uns von falschen Theorien täuschen lassen*. Berlin University Press. <https://books.google.de/books?id=HWUKaAEACAAJ>.

Kuhn, Max, und Kjell Johnson. 2013. *Applied predictive modeling*. Bd. 26. Springer.

Ligges, Uwe, Martin Maechler, und Sarah Schnackenberg. 2017. *scatterplot3d: 3D Scatter Plot*. <https://CRAN.R-project.org/package=scatterplot3d>.

Milborrow, Stephen. 2017. *rpart.plot: Plot 'rpart' Models: An Enhanced Version of 'plot.rpart'*. <https://CRAN.R-project.org/package=rpart.plot>.

Moore, David S. 1990. „Uncertainty“. *On the shoulders of giants: New approaches to numeracy*. ERIC, 95–137.

Mullen, Lincoln. 2016. *tokenizers: A Consistent Interface to Tokenize Natural Language Text*. <https://CRAN.R-project.org/package=tokenizers>.

Neuwirth, Erich. 2014. *RColorBrewer: ColorBrewer Palettes*. <https://CRAN.R-project.org/package=RColorBrewer>.

Ooms, Jeroen. 2016. *pdftools: Text Extraction and Rendering of PDF Documents*. <https://CRAN.R-project.org/package=pdftools>.

Peirce, Charles S. 1955. „Abduction and induction“. *Philosophical writings of Peirce* 11. New York.

Peng, Roger D, und Elizabeth Matsui. 2015. „The Art of Data Science“. *A Guide for Anyone Who Works with Data*. Skybrude Consulting 200: 162.

Raiche, Gilles, und David Magis. 2011. *nFactors: Parallel Analysis and Non Graphical Solutions to the Cattell Scree Test*. <https://CRAN.R-project.org/package=nFactors>.

Ram, Karthik, und Hadley Wickham. 2015. *wesanderson: A Wes Anderson Palette Generator*. <https://CRAN.R-project.org/package=wesanderson>.

Re, AC Del. 2014. *compute.es: Compute Effect Sizes*. <https://CRAN.R-project.org/>

[package=compute.es.](#)

Remus, R., U. Quasthoff, und G. Heyer. 2010. „SentiWS – a Publicly Available German-language Resource for Sentiment Analysis“. In *Proceedings of the 7th International Language Resources and Evaluation (LREC'10)*, 1168–71.

Ripley, Brian. 2016. *MASS: Support Functions and Datasets for Venables and Ripley's MASS*. <https://CRAN.R-project.org/package=MASS>.

RITA, Bureau of transportation statistics. 2013. „nycflights13“. [http://www.transtats.bts.gov/DL{\\\_}Select](http://www.transtats.bts.gov/DL{\_}Select)

Robinson, David. 2016. *gutenbergr: Download and Process Public Domain Works from Project Gutenberg*. <https://cran.rstudio.com/package=gutenbergr>.

Robinson, David, Matthieu Gomez, Boris Demeshev, Dieter Menne, Benjamin Nutter, Luke Johnston, Ben Bolker, Francois Briatte, und Hadley Wickham. 2015. *broom: Convert Statistical Analysis Objects into Tidy Data Frames*. <https://CRAN.R-project.org/package=broom>.

Robinson, David, und Julia Silge. 2016. *tidytext: Text Mining using 'dplyr', 'ggplot2', and Other Tidy Tools*. <https://CRAN.R-project.org/package=tidytext>.

Romeijn, Jan-Willem. 2016. „Philosophy of Statistics“. In *The Stanford Encyclopedia of Philosophy*, herausgegeben von Edward N. Zalta, Winter 2016. <http://plato.stanford.edu/archives/win2016/entries/statistics/>.

Rucker, R. o. J. *Infinity and the Mind*. ?????? ?????? <https://books.google.de/books?id=MDOUAWAAQBAJ>.

Sauer, Sebastian. 2016. „Extraversion Dataset“. Open Science Framework. doi:10.17605/OSF.IO/4KGZH<sup>20</sup>.

———. 2017a. „Dataset ‚predictors of performance in stats test‘“. Open Science Framework. doi:10.17605/OSF.IO/SJHUY<sup>21</sup>.

———. 2017b. „Dataset ‚Height and shoe size‘“. Open Science Framework. doi:10.17605/OSF.IO/JA9DW<sup>22</sup>.

Sauer, Sebastian, und Alexander Wolff. 2016. „The effect of a status symbol on success in online dating: an experimental study (data paper)“. *The Winnower*, August. doi:10.15200/winn.147241.13309<sup>23</sup>.

Schloerke, Barret, Jason Crowley, Di Cook, Francois Briatte, Moritz Marbach, Edwin Thoen, Amos Elberg, und Joseph Larmarange. 2016. *GGally: Extension to 'ggplot2'*. <https://CRAN.R-project.org/package=GGally>.

Silge, Julia. 2016. *janeaustenr: Jane Austen's Complete Novels*. <https://CRAN.R-project.org/package=janeaustenr>.

Silge, Julia, David Robinson, und Jim Hester. 2016. „tidytext: Text mining using dplyr,

<sup>19</sup>[http://www.transtats.bts.gov/DL%7B/\\_%7DSelectFields.asp?Table%7B/\\_%7DID=236](http://www.transtats.bts.gov/DL%7B/_%7DSelectFields.asp?Table%7B/_%7DID=236)

<sup>20</sup><https://doi.org/10.17605/OSF.IO/4KGZH>

<sup>21</sup><https://doi.org/10.17605/OSF.IO/SJHUY>

<sup>22</sup><https://doi.org/10.17605/OSF.IO/JA9DW>

<sup>23</sup><https://doi.org/10.15200/winn.147241.13309>

ggplot2, and other tidy tools“. doi:10.5281/zenodo.56714<sup>24</sup>.

Silge, Julia, und David Robinson. 2016. „tidytext: Text Mining and Analysis Using Tidy Data Principles in R“. *The Journal of Open Source Software* 1 (3). The Open Journal. doi:10.21105/joss.00037<sup>25</sup>.

Suppes, Patrick, und Joseph L Zinnes. 1962. *Basic measurement theory*. Institute for mathematical studies in the social sciences.

*The Oxford Dictionary of Statistical Terms*. 2006. Oxford University Press. <https://www.amazon.com/Oxford-Dictionary-Statistical-Terms/dp/0199206139%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0199206139>.

Therneau, Terry, Beth Atkinson, und Brian Ripley. 2015. *rpart: Recursive Partitioning and Regression Trees*. <https://CRAN.R-project.org/package=rpart>.

Tufte, Edward R. 1990. *Envisioning Information*. Graphics Press. <https://www.amazon.com/Envisioning-Information-Edward-R-Tufte/dp/1930824149%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1930824149>.

———. 2001. *The Visual Display of Quantitative Information*. Graphics Press. <https://www.amazon.com/Visual-Display-Quantitative-Information/dp/1930824130%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1930824130>.

———. 2006. *Beautiful Evidence*. Graphics Press. <https://www.amazon.com/Beautiful-Evidence-Edward-R-Tufte/dp/1930824165%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1930824165>.

VanDerWal, Jeremy, Lorena Falconi, Stephanie Januchowski, Luke Shoo, und Collin Storlie. 2014. *SDMTools: Species Distribution Modelling Tools: Tools for processing data associated with species distribution modelling exercises*. <https://CRAN.R-project.org/package=SDMTools>.

Wagenmakers, Eric-Jan. 2007. „A practical solution to the pervasive problems of values“. *Psychonomic Bulletin & Review* 14 (5). Springer Nature: 779–804. doi:10.3758/bf03194105<sup>26</sup>.

Warnes, Gregory R., Ben Bolker, Lodewijk Bonebakker, Robert Gentleman, Wolfgang Huber, Andy Liaw, Thomas Lumley, Martin Maechler, u. a. 2016. *gplots: Various R Programming Tools for Plotting Data*. <https://CRAN.R-project.org/package=gplots>.

Wei, Taiyun, und Viliam Simko. 2016. *corrplot: Visualization of a Correlation Matrix*. <https://CRAN.R-project.org/package=corrplot>.

Wicherts, Jelte M., Coosje L. S. Veldkamp, Hilde E. M. Augusteijn, Marjan Bakker, Robbie

<sup>24</sup><https://doi.org/10.5281/zenodo.56714>

<sup>25</sup><https://doi.org/10.21105/joss.00037>

<sup>26</sup><https://doi.org/10.3758/bf03194105>

- C. M. van Aert, und Marcel A. L. M. van Assen. 2016. „Degrees of Freedom in Planning, Running, Analyzing, and Reporting Psychological Studies: A Checklist to Avoid p-Hacking“. *Frontiers in Psychology* 7 (November). Frontiers Media SA. doi:10.3389/fpsyg.2016.01832<sup>27</sup>.
- Wickham, Hadley. 2009. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. <http://ggplot2.org>.
- . 2014. „Tidy Data“. *Journal of Statistical Software* 59 (1): 1–23. doi:10.18637/jss.v059.i10<sup>28</sup>.
- . 2016a. *reshape2: Flexibly Reshape Data: A Reboot of the Reshape Package*. <https://CRAN.R-project.org/package=reshape2>.
- . 2016b. *tidyr: Easily Tidy Data with ‘spread()’ and ‘gather()’ Functions*. <https://CRAN.R-project.org/package=tidyr>.
- . 2017a. *nycflights13: Flights that Departed NYC in 2013*. <https://CRAN.R-project.org/package=nycflights13>.
- . 2017b. *stringr: Simple, Consistent Wrappers for Common String Operations*. <https://CRAN.R-project.org/package=stringr>.
- . 2017c. *tidyverse: Easily Install and Load ‘Tidyverse’ Packages*. <https://CRAN.R-project.org/package=tidyverse>.
- Wickham, Hadley, Jim Hester, und Romain Francois. 2016a. *readr: Read Tabular Data*. <https://CRAN.R-project.org/package=readr>.
- . 2016b. *readr: Read Tabular Data*. <https://CRAN.R-project.org/package=readr>.
- Wickham, Hadley, und Romain Francois. 2016. *dplyr: A Grammar of Data Manipulation*. <https://CRAN.R-project.org/package=dplyr>.
- Wickham, Hadley, und Garrett Golemund. 2016. *R for Data Science: Visualize, Model, Transform, Tidy, and Import Data*. O’Reilly Media. <https://www.amazon.com/Data-Science-Visualize-Model-Transform/dp/1491910399%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1491910399>.
- Wild, Chris J, und Maxine Pfannkuch. 1999. „Statistical thinking in empirical enquiry“. *International Statistical Review* 67 (3). Wiley Online Library: 223–48.
- Wild, Fridolin. 2015. *lsa: Latent Semantic Analysis*. <https://CRAN.R-project.org/package=lsa>.
- Wilkinson, Leland. 2006. *The grammar of graphics*. Springer Science & Business Media.
- Xie, Yihui. 2015. *Dynamic Documents with R and knitr*. 2nd Aufl. Boca Raton, Florida: Chapman; Hall/CRC. <http://yihui.name/knitr/>.
- . 2016. *knitr: A General-Purpose Package for Dynamic Report Generation in R*. <https://CRAN.R-project.org/package=knitr>.

<sup>27</sup><https://doi.org/10.3389/fpsyg.2016.01832>

<sup>28</sup><https://doi.org/10.18637/jss.v059.i10>

[//CRAN.R-project.org/package=knitr](https://CRAN.R-project.org/package=knitr).

Zumel, Nina, John Mount, und Jim Porzak. 2014. *Practical data science with R*. Manning.

# Index

Datenjudo, [viii](#)  
dplyr::arrange, [xiv](#)  
dplyr::count, [xxiv](#)  
dplyr::filter, [x](#)  
dplyr::mutate, [xxviii](#)  
dplyr::n, [xxiii](#)  
dplyr::select, [xii](#)  
dplyr::summarise, [xix](#)

Lagemaße, [xxi](#)

Pfeife, [xxv](#), [xxvi](#)

Streuungsmaße, [xxii](#)