

# Einführung in R

## Teil 1.1

Linda T. Betz, MSc  
Kambeitz Lab  
University Hospital Cologne  
April 1, 2020



**KAMBEITZ LAB**

Prediction and Prevention in  
Mental Health

# Kurze Vorstellungsrunde

- Wie heißt du und was ist dein Hintergrund?
- Wofür planst du, R zu verwenden?
- Hast du Vorkenntnisse in R oder einer anderen Programmiersprache?

# Warum R?

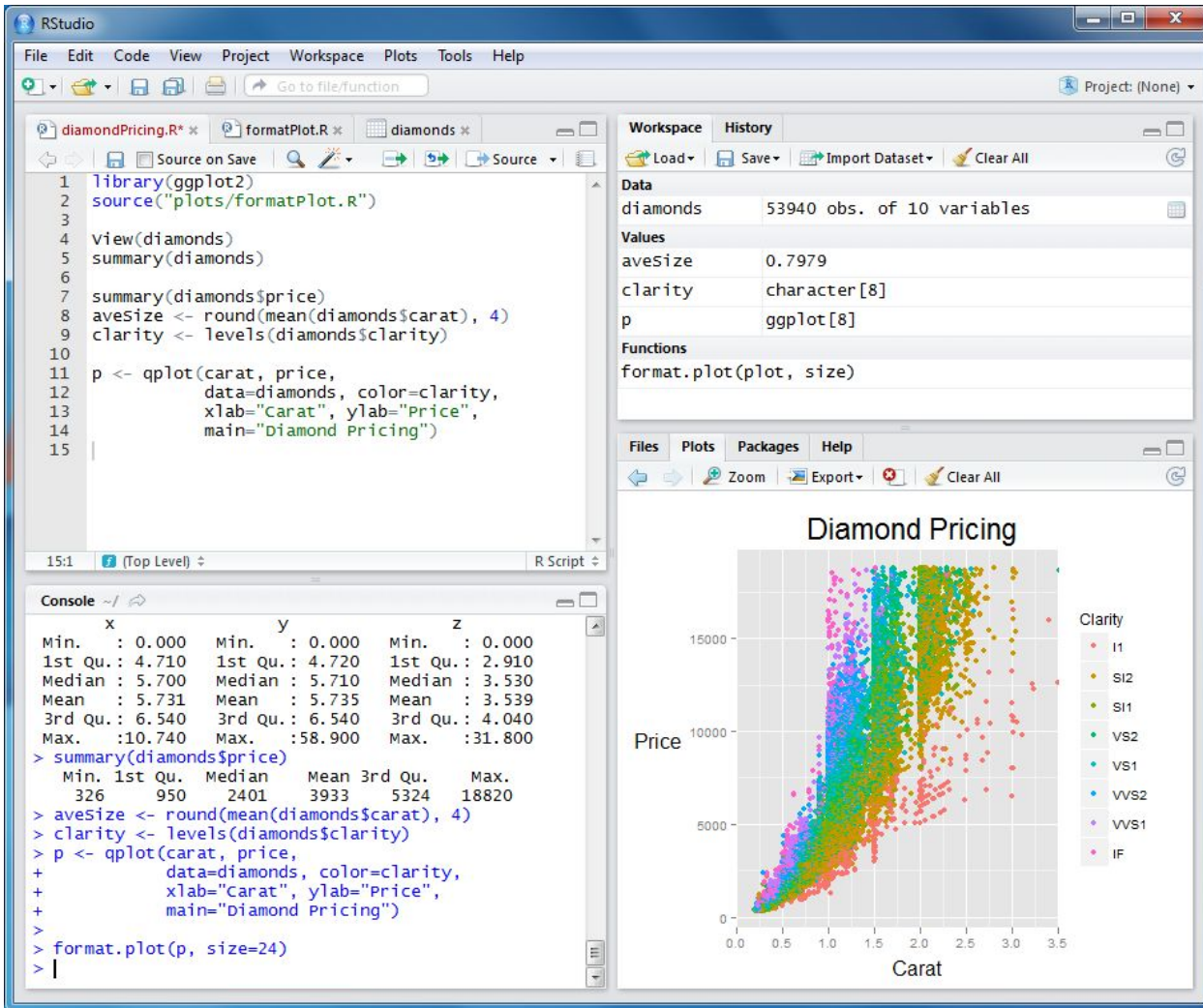
- Optimiert für Datenanalyse & Visualisierung
- “open source” → Zugriff auf Code: anders als z.B. in SPSS ist immer ganz klar, was in einer Funktion passiert
  - Abänderungen leicht möglich
- Viele Packages für fortgeschrittene, komplexe Statistik
- Sehr aktive Community → Unterstützung bei Problemen, Teilen von Code



# TEIL I: Die Basics

(hauptsächlich basierend auf <https://r4ds.had.co.nz/>)

Code →



← Workspace  
("Arbeits-  
speicher")

← Plots, Hilfe,  
Files...

# Die Basics

R als “Taschenrechner”:

```
1 / 200 * 30  
#> [1] 0.15  
(59 + 73 + 2) / 3  
#> [1] 44.7  
sin(pi / 2)  
#> [1] 1
```

# Die Basics

Ein neues Objekt erstellen mit “<-”:

```
x <- 3 * 4
```

```
object_name <- value
```

“Objekt erhält Wert...”

Nicht faul sein! Obwohl “=” die gleiche Funktion hat, kann es zu Verwirrung führen.  
Shortcut für “<-”: Alt + -

# Die Basics

Objektnamen...

... müssen mit einem Buchstaben beginnen

... dürfen keine anderen Zeichen als Buchstaben, Zahlen, “\_”, und “.” enthalten

Good Practice: “snake\_case”

```
i_use_snake_case  
otherPeopleUseCamelCase  
some.people.use.periods  
And_aFew.People_RENOUNCEconvention
```



# Die Basics

Wir können ein Objekt genauer anschauen, wenn wir einfach den Namen in die R Console eintippen:

```
x  
#> [1] 12
```

# Die Basics

Wir können beliebig lange Variablennamen verwenden:

```
this_is_a_really_long_name <- 2.5
```

Wenn wir uns vertan haben und z.B. eigentlich den Wert 3.5 setzen wollten, können wir das einfach tun, indem wir 2.5 überschreiben.

# Die Basics

Wir können ein neues Beispiel erstellen - “^” bedeutet “hoch”, also  $2^3$ :

```
r_rocks <- 2 ^ 3
```

```
r_rock
```

```
#> Error: object 'r_rock' not found
```

```
R_rocks
```

```
#> Error: object 'R_rocks' not found
```

# Die Basics

```
function_name(arg1 = val1, arg2 = val2, ...)
```

```
seq(1, 10)
```

```
#> [1] 1 2 3 4 5 6 7 8 9 10
```

```
y <- seq(1, 10, length.out = 5)
```

```
y
```

```
#> [1] 1.00 3.25 5.50 7.75 10.00
```

# Die Basics

```
seq(1, 10)  
#> [1] 1 2 3 4 5 6 7 8 9 10
```

Eine genaue Beschreibung dessen, was eine Funktion macht und welche Inputargumente sie benötigt, findet man über die Hilfe (“Help”)-Seiten:

```
?seq
```

# Sequence Generation

## Description

Generate regular sequences. `seq` is a standard generic with a default method. `seq.int` is a primitive which can be much faster but has a few restrictions. `seq_along` and `seq_len` are very fast primitives for two common cases.

## Usage

```
seq(...)  
  
## Default S3 method:  
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),  
    length.out = NULL, along.with = NULL, ...)  
  
seq.int(from, to, by, length.out, along.with, ...)  
  
seq_along(along.with)  
seq_len(length.out)
```

## Arguments

<code>...</code>	arguments passed to or from methods.
<code>from</code> , <code>to</code>	the starting and (maximal) end values of the sequence. Of length 1 unless just <code>from</code> is supplied as an unnamed argument.
<code>by</code>	number: increment of the sequence.
<code>length.out</code>	desired length of the sequence. A non-negative number, which for <code>seq</code> and <code>seq.int</code> will be rounded up if fractional.
<code>along.with</code>	take the length from the length of this argument.

## Details

Numerical inputs should all be [finite](#) (that is, not infinite, [NaN](#) or [NA](#)).

The interpretation of the unnamed arguments of `seq` and `seq.int` is *not* standard, and it is recommended always to name the arguments when programming.

# Die Basics



Allgemein wichtig:

Erfahrungsgemäß lassen sich 99% aller Probleme / Fehlermeldung in R erfolgreich durch Googlen lösen!

Strategien:

- Fehlermeldung kopieren (am besten auf Englisch, R auf Englisch benutzen)
- **auf Englisch suchen**
- guter Startpunkt: Stackoverflow

# Die Basics

```
x <- "hello world"
```

```
> x <- "hello  
+
```

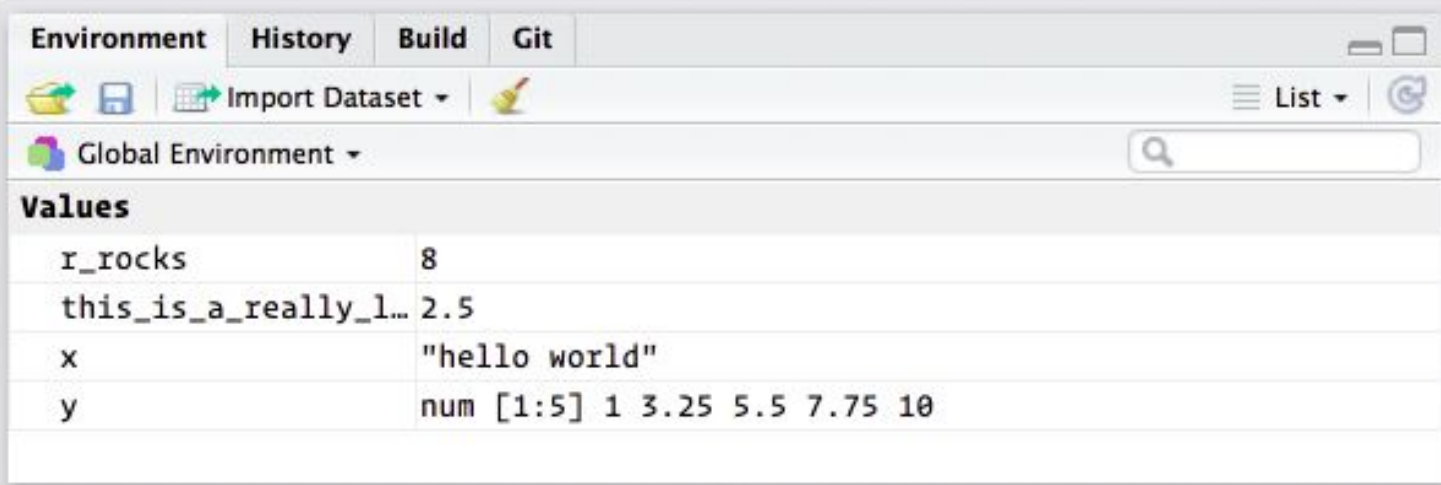
“+” bedeutet, dass R auf weiteren Input wartet. Hier: ein abschließendes **”**.

“” und () kommen immer paarweise!



# Die Basics

In unserem Workspace sind nun folgende Objekte gespeichert:



Environment		History	Build	Git
Import Dataset		List		
Global Environment				
Values				
r_rocks	8			
this_is_a_really_l...	2.5			
x	"hello world"			
y	num [1:5] 1 3.25 5.5 7.75 10			

# Übung

Warum funktioniert das so nicht?

```
my_variable <- 10  
my_variable  
#> Error in eval(expr, envir, enclos): object 'my_variable' not found
```

R ist (wie meisten Programmiersprachen) “case-sensitive”, das heißt, wir müssen ganz genau sein bezüglich der Schreibweise (Groß- und Kleinschreibung, etc.)

# Installation von Packages

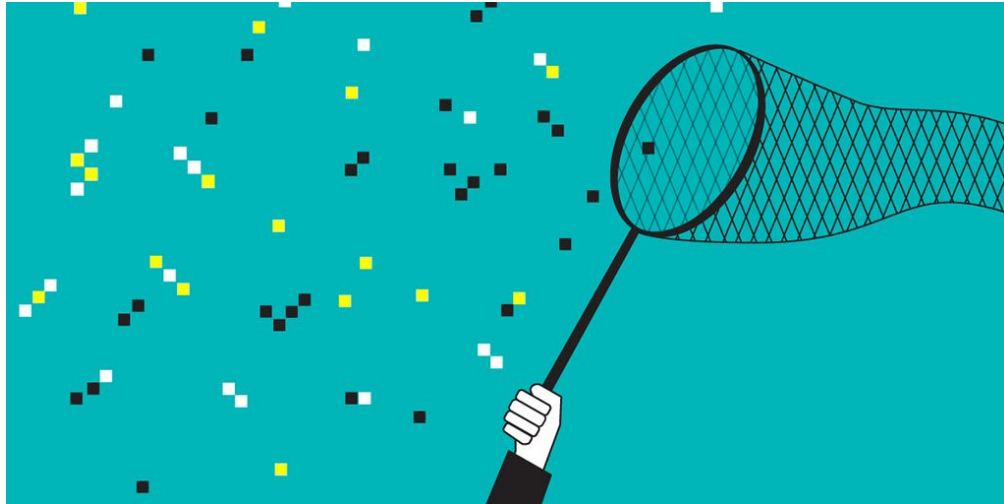
Viele Funktionen, die wir verwenden, sind nicht direkt in R verfügbar, sondern müssen erst über Erweiterungen, sog. “packages”, geladen werden.

Notwendig ist dazu die Installation über

```
install.packages("ncyflights13")  
install.packages("tidyverse")
```

# TEIL II: Datentransformation

(hauptsächlich basierend auf <https://r4ds.had.co.nz/>)



# Warum Datentransformation?

Man bekommt die Daten selten genau so, wie man sie braucht...

???

Mögliche Probleme:

- unklare Variablennamen
- fehlende Werte als Zahlen kodiert (z.B. "-999")
- Skalenwerte müssen erst noch berechnet werden
- Leute mit zu vielen fehlenden Daten sollen ausgeschlossen werden

```
sitecode site country crfnr rr001_dat landcode a002 a003 a007 a008
<dbl> <chr> <chr> <dbl> <date> <dbl> <chr> <dbl> <dbl> <dbl> <dbl>
1 7 Shlv~ Israel 1 2003-08-04 972 G P 2 [Fem~ 1 [yes] 1 [yes] 1
2 7 Shlv~ Israel 2 2003-10-13 972 A Y 1 [Mal~ 1 [yes] 1 [yes] 1
3 6 Beer~ Israel 3 2003-02-03 972 ETS 1 [Mal~ 1 [yes] 1 [yes] 1
4 6 Beer~ Israel 4 2003-02-09 972 R M 2 [Fem~ 1 [yes] 1 [yes] 1
5 5 Sheb~ Israel 5 2002-12-23 972 R R 2 [Fem~ 1 [yes] 1 [yes] 1
6 5 Sheb~ Israel 6 2002-12-23 972 T M 2 [Fem~ 1 [yes] 1 [yes] 1
7 5 Sheb~ Israel 7 2003-01-13 972 A A 1 [Mal~ 1 [yes] 1 [yes] 1
8 5 Sheb~ Israel 8 2003-06-18 972 E K 1 [Mal~ 1 [yes] 1 [yes] 1
9 6 Beer~ Israel 9 2003-03-17 972 EAs 2 [Fem~ 1 [yes] 1 [yes] 1
10 6 Beer~ Israel 10 2003-04-09 972 BAD 2 [Fem~ 1 [yes] 1 [yes] 1
# with 488 more rows and 72 more variables: a074 <dbl>, a075 <dbl>, a076 <dbl>
```

```
v1_id v1_gaf
1 aagd330 65
2 aaph286 55
3 aaru067 61
4 ablx139 78
5 abmk123 89
6 abuj304 75
7 achw003 -999
8 acok454 89
9 adrr818 55
10 adsm526 NA
11 adxo842 NA
12 aeea226 NA
13 aefl766 60
14 afsj906 50
15 aghk530 60
16 agts319 -999
17 ahjl311 51
18 ainp421 45
19 ajfy391 66
20 ajge751 59
```

not  
good...

# Datentransformation

Für Datentransformationen verwenden wir das R package *tidyverse*. Grundkonzepte daraus werden wir uns anhand von Flugdaten von New York aus dem Jahr 2013 ansehen. Dazu laden wir die Packages mit der Funktion “library”:

```
library(nycflights13)  
library(tidyverse)
```

Achtet auf die Nachrichten, die ausgegeben werden, wenn man die Packages lädt. Es zeigt an, dass dplyr einige Funktionen aus “base R” überschreibt; diese müssen jetzt mit ihrem vollen Namen aufgerufen werden (z.B. stats::filter).

# Datentransformation

Wir sehen uns zunächst den Datensatz an, mit dem wir arbeiten werden. Dazu tippen wir `flights`

336,776 Zeilen:

Beobachtungen

19 Spalten:

Variablen

“Zeile mal Spalte”

Aktuell zeigt uns R nur einen Teil der Daten. Um den gesamten Datensatz sehen, tippt man:

`View(flights)`

```
flights
#> # A tibble: 336,776 x 19
#>   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>   <int> <int> <int>   <int>         <int>       <dbl>   <int>         <int>
#> 1  2013     1     1     517           515         2     830           819
#> 2  2013     1     1     533           529         4     850           830
#> 3  2013     1     1     542           540         2     923           850
#> 4  2013     1     1     544           545        -1    1004          1022
#> 5  2013     1     1     554           600        -6     812           837
#> 6  2013     1     1     554           558        -4     740           728
#> # ... with 3.368e+05 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
```

# Datentransformation

Kurz zum Datensatz: in `flights` sind alle Abflüge von den 3 New Yorker Flughäfen (JFK, LGA, EWR) im Jahr 2013 gespeichert

Jede Zeile repräsentiert einen Flug

Jede Spalte eine Variable, die einen Aspekt zu den Flügen misst (Airline, Verspätung, etc.). Mehr Infos zu den Variablen findet sich über: `?flights`

Quelle: RITA, Bureau of Transportation Statistics





# Datentransformation

Variablen in R können verschiedene Typen haben.

- `int` steht für **Integer**, also ganze Zahlen (1,2,...).
- `dbl` steht für **Doubles**, also reelle Zahlen (0.25, 1.00, -5)
- `chr` steht für **Character**/Strings (Buchstaben, "abc")
- `dtm` steht für **Date-Time** (Datum plus Uhrzeit)

Es gibt noch 3 weitere Typen, die hier nicht vorkommen:

- `lgl` steht für **Logical**, Vektoren die nur TRUE oder FALSE enthalten (Beispiel: "Psychose")
- `fctr` steht für **Faktoren**, die verwendet werden, um kategoriale Variablen mit einer klar umschriebenen Anzahl an möglichen Werten darzustellen (z.B. Haarfarbe: "blond"/"braun"/"sonstiges")
- `date` steht für **Daten** (z.B. 31.12.2013)

```
flights
#> # A tibble: 336,776 x 19
#>   year month   day dep_time sched_dep_time dep_delay arr_time
#>   <int> <int> <int>   <int>         <int>    <dbl>   <int>
#> 1  2013     1     1     517           515         2     830
#> 2  2013     1     1     533           529         4     850
#> 3  2013     1     1     542           540         2     923
#> 4  2013     1     1     544           545        -1    1004
#> 5  2013     1     1     554           600        -6     812
#> 6  2013     1     1     554           558        -4     740
#> # ... with 3.368e+05 more rows, and 11 more variables: arr_delay,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, a
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, t
```

# Datentransformation

Oft ist es hilfreich, sich nach Reinladen der Daten erst mal einen groben Überblick zu verschaffen. Das hilft, mögliche Fehler schnell zu entdecken. Dazu gibt es verschiedene Möglichkeiten:

`head(flights)` # => zeigt die ersten 6 Zeilen des Datensatzes

`str(flights)` # => zeigt, welche Variablentypen enthalten sind

`flights$carrier` # => erlaubt, sich eine einzelne Variable anzusehen

`flights[,1]` # => erste Spalte des Datensatzes

`flights[1,]` # => erste Zeile des Datensatzes

`flights[1,1]` # => erste Zeile, erste Spalte des Datensatzes

`summary(flights)` # => gibt Summary-Statistics für jede Variable (Mittelwert, Median, fehlende Werte...) 26

# Datentransformation

- Auswahl von Beobachtungen anhand ihres Wertes mit `filter()`.
- Umordnen von Beobachtungen mit `arrange()`.
- Auswahl von Variablen mit `select()`.
- Erzeugen neuer Variablen anhand existierender Variablen mit `mutate()`.
- Gruppieren des Datensatzes anhand einer Variable mit `group_by()`.
- Zusammenfassen vieler Werte in einen Wert mit `summarise()`.

Alle diese “Verben” funktionieren auf dieselbe Weise:

1. Das erste Argument ist eine *data frame* (Objekt, in dem die Daten gespeichert sind).
2. Die folgenden Argumente sind “Verben”, die anzeigen, was mit den Daten passieren soll.
3. Das Ergebnis ist eine neue *data frame*.

# Datentransformation

Reihen filtern mit `filter()`. Das erste Argument ist die *data frame*, die gefiltert werden soll. Das zweite und nachfolgende Argumente zeigen, wonach gefiltert werden soll.

Wir können uns z.B. alle Flüge ausgeben lassen, die am 1. Januar 2013 gestartet sind:

```
filter(flights, month == 1, day == 1)

#> # A tibble: 842 x 19
#>   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>   <int> <int> <int>   <int>         <int>      <dbl>    <int>         <int>
#> 1  2013     1     1     517           515         2      830           819
#> 2  2013     1     1     533           529         4      850           830
#> 3  2013     1     1     542           540         2      923           850
#> 4  2013     1     1     544           545        -1     1004          1022
#> 5  2013     1     1     554           600        -6      812           837
#> 6  2013     1     1     554           558        -4      740           728
#> # ... with 836 more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#> #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
```

# Datentransformation

Wichtig: *tidyverse* verändert seinen Input NIE direkt. Das heißt, wir müssen **neue** Ergebnisse abspeichern mit “<-”:

```
jan1 <- filter(flights, month == 1, day == 1)
```

# Datentransformation

Um `filter()` effektiv nutzen zu können, kann man in R, wie in anderen Programmiersprachen, auf diese Möglichkeiten zugreifen:

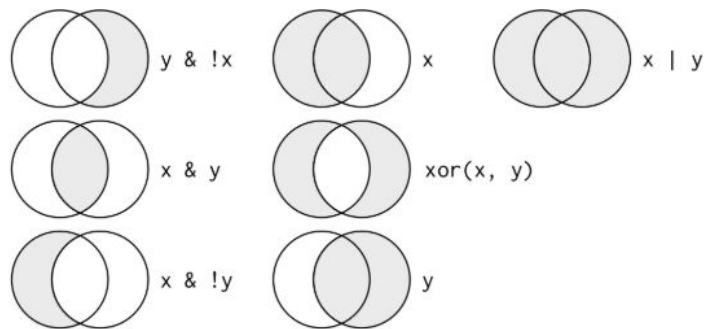
`>`, `>=`, `<`, `<=`, `!=` (not equal), and `==` (equal).

Ein typischer Fehler gerade am Anfang ist, “=” anstatt “==” zu verwenden:

```
filter(flights, month = 1)
#> Error: `month` (`month = 1`) must not be named, do you need `==`?
```

# Datentransformation

Auch erlaubt `filter()`, verschiedene Argumente zu kombinieren. Dazu verwendet man logische Operatoren. `&` bedeutet “UND”, `|` bedeutet “ODER”, und `!` bedeutet “NICHT”.



```
filter(flights, month == 11 | month == 12)
```

```
nov_dec <- filter(flights, month %in% c(11, 12))
```

Flüge, die im November oder  
Dezember 2013 abflogen

# Datentransformation

Etwas komplexere Filteroperationen:

```
filter(flights, !(arr_delay > 120 | dep_delay > 120))  
filter(flights, arr_delay <= 120, dep_delay <= 120)
```

Was bedeuten diese Zeilen?

Wir filtern Flüge, die weder bei Abflug noch bei Ankunft mehr als 2 Stunden Verspätung hatten.



# Datentransformation

## Fehlende Werte

**NA** steht in R dafür, dass ein Wert fehlt. Fast jede Operation, die auf Daten ausgeführt wird, die fehlende Werte enthalten, gibt NA zurück:

```
NA > 5
#> [1] NA

10 == NA
#> [1] NA

NA + 10
#> [1] NA

NA / 2
#> [1] NA
```

# Datentransformation

Sehr verwirrend kann folgendes Verhalten sein:

```
NA == NA  
#> [1] NA
```

Angenommen, wir wollen das Alter zweier Personen vergleichen.

- Pedros Alter: `x <- NA`
- Noras Alter: `y <- NA`
- In beiden Fällen zeigt uns NA, dass wir nicht wissen, wie alt Pedro und Nora sind. Deswegen liefert `x == y` ebenfalls NA - wir haben schlicht zu wenig Information für den Vergleich (sie könnten gleich alt sein oder nicht).

# Datentransformation

Um zu bestimmen, ob ein Wert fehlt, verwenden wir `is.na()`

```
is.na(x)
```

```
#> [1] TRUE
```

PAUSE! 5 min :)



# Übungen (Übung A)

Finde alle Flüge, die...

1. bei Ankunft sechs oder mehr Stunden verspätet waren.
2. nach San Antonio, Texas flogen (SAT). Wie viele waren es?
3. im Dezember nach San Antonio flogen. Wie viele waren es?
4. nach Houston, Texas flogen (IAH oder HOU)
5. von den Fluglinien American (AA) oder Delta (DL) durchgeführt wurden (Variable "carrier")
6. im Sommer abflogen (Sommer = Juli, August, September)
7. Wie viele Flüge haben einen fehlenden Wert (NA) in der Variable dep\_time?
8. BONUS: mindestens eine Stunde verspätet waren, aber im Flug über 30 Minuten aufholten
9. BONUS: Warum ergibt `NA ^ 0` nicht NA? Warum ist `NA | TRUE` nicht NA?



# Datentransformation

Zeilen umordnen mit `arrange()`

`arrange()` funktioniert ähnlich wie `filter()`, nur, dass wir jetzt die Reihenfolge der Zeilen verändern.

```
arrange(flights, year, month, day)
```

Wir verwenden “desc”, um eine Spalte in absteigender Reihenfolge zu ordnen:

```
arrange(flights, desc(dep_delay))
```

# Übungen (Übung B)

1. Sortiere nach den Flügen mit der größten Verspätung bei Ankunft.
2. Welcher Flug war bezogen auf die Entfernung der kürzeste?
3. BONUS: sortiere die Flüge nach San Antonio in Bezug auf Verspätung bei Ankunft.
4. BONUS: Wie könnte man `arrange()` verwenden, um alle “NAs” in einer Variable an den Anfang zu schieben? (Tipp: verwende `is.na()`).



# Datentransformation

Auswahl von Variablen mit `select()`

Es ist nicht unüblich, dass man Datensätze mit mehreren hundert Variablen bekommt.

Dann macht es Sinn, die Auswahl einzuschränken.

```
# Select columns by name
select(flights, year, month, day)

#> # A tibble: 336,776 x 3
#>   year month   day
#>   <int> <int> <int>
#> 1  2013     1     1
#> 2  2013     1     1
#> 3  2013     1     1
#> 4  2013     1     1
#> 5  2013     1     1
#> 6  2013     1     1
#> # ... with 3.368e+05 more rows
```



# Datentransformation

Für eine Auswahl der Variablen  
zwischen “year” und “day”

```
select(flights, year:day)
#> # A tibble: 336,776 x 3
#>   year month   day
#>   <int> <int> <int>
#> 1  2013     1     1
#> 2  2013     1     1
#> 3  2013     1     1
#> 4  2013     1     1
#> 5  2013     1     1
#> 6  2013     1     1
#> # ... with 3.368e+05 more rows
```

# Datentransformation

Für eine Auswahl aller Variablen *außer* denen zwischen “year” und “day”

```
select(flights, -(year:day))
#> # A tibble: 336,776 x 16
#>   dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
#>   <int>         <int>      <dbl>   <int>         <int>        <dbl> <chr>
#> 1     517           515         2     830           819         11 UA
#> 2     533           529         4     850           830         20 UA
#> 3     542           540         2     923           850         33 AA
#> 4     544           545        -1    1004          1022        -18 B6
#> 5     554           600        -6     812           837        -25 DL
#> 6     554           558        -4     740           728         12 UA
#> # ... with 3.368e+05 more rows, and 9 more variables: flight <int>,
#> #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#> #   hour <dbl>, minute <dbl>, time_hour <dtm>
```

# Datentransformation

Wichtige “Helferfunktionen” für `select()`:

- `starts_with("dep")`: Namen, die mit “dep” beginnen.
- `ends_with("time")`: Namen, die mit “time” enden
- `contains("air")`: Namen, die “air” enthalten
- `matches("(.)\\1")`: Namensauswahl mit regular expressions. Sehr nützlich!
- `num_range("x", 1:3)`: Namen x1, x2, und x3

# Datentransformation

`Select()` + `everything()` helper. Hilft, wenn man bestimmte Variablen an den Anfang der data frame schieben will.

```
select(flights, time_hour, air_time, everything())  
#> # A tibble: 336,776 x 19  
#>   time_hour          air_time  year month   day dep_time sched_dep_time  
#>   <dtm>              <dbl> <int> <int> <int>   <int>         <int>  
#> 1 2013-01-01 05:00:00    227  2013     1     1     517           515  
#> 2 2013-01-01 05:00:00    227  2013     1     1     533           529  
#> 3 2013-01-01 05:00:00    160  2013     1     1     542           540  
#> 4 2013-01-01 05:00:00    183  2013     1     1     544           545
```

# Datentransformation

`rename()` (Variante von `select()`) zum Umbenennen von Variablen:

```
rename(flights, tail_num = tailnum)
#> # A tibble: 336,776 x 19
#>   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>   <int> <int> <int>   <int>         <int>       <dbl>   <int>         <int>
#> 1  2013     1     1     517           515         2     830           819
#> 2  2013     1     1     533           529         4     850           830
#> 3  2013     1     1     542           540         2     923           850
#> 4  2013     1     1     544           545        -1    1004          1022
#> 5  2013     1     1     554           600        -6     812           837
#> 6  2013     1     1     554           558        -4     740           728
#> # ... with 3.368e+05 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tail_num <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
```

# Übungen (Übung C)

1. Benenne die Variablen “year”, “month”, und “day” in deutsche Begriffe um.
2. Wähle alle Variablen aus, die das Wort “time” enthalten.
3. BONUS: Warum funktioniert `flights %>% filter(gain, speed)` nicht? Wofür verwendet man `filter()` im Gegensatz zu `select()`?



# Datentransformation

Was passiert hier?

```
flights_sml <- select(flights,  
  year:day,  
  ends_with("delay"),  
  distance,  
  air_time  
)
```

# Datentransformation

Neue Variablen generieren mit `mutate()`

```
mutate(flights_sml,  
  gain = dep_delay - arr_delay,  
  speed = distance / air_time * 60  
)
```

```
#> # A tibble: 336,776 x 9
```

```
#>   year month   day dep_delay arr_delay distance air_time  gain speed  
#>   <int> <int> <int>     <dbl>     <dbl>     <dbl>     <dbl> <dbl> <dbl>  
#> 1  2013     1     1         2         11    1400      227    -9   370.  
#> 2  2013     1     1         4         20    1416      227   -16   374.  
#> 3  2013     1     1         2         33    1089      160   -31   408.
```



# Datentransformation

Wenn man nur die neu erzeugten Variablen behalten will, verwendet man `transmute()`

```
transmute(flights,  
  gain = dep_delay - arr_delay,  
  hours = air_time / 60,  
  gain_per_hour = gain / hours  
)  
  
#> # A tibble: 336,776 x 3  
  
#>   gain hours gain_per_hour  
#>   <dbl> <dbl>         <dbl>  
#> 1     -9  3.78         -2.38  
#> 2    -16  3.78         -4.23  
#> 3    -31  2.67        -11.6
```

# Datentransformation

## Nützliche Operatoren

- `+`, `-`, `*`, `/`, `^`: `air_time / 60, hours * 60 + minute`, etc.
  - a. `x / sum(x)` berechnet einen Anteil vom Ganzen
  - b. `y - mean(y)` berechnet die Differenz zum Mittelwert
- Logs: `log()`, `log2()`, `log10()`.
- Logical comparisons, `<`, `<=`, `>`, `>=`, `!=`, and `==`,
- Reihung/Reihenfolge: `min_rank()`.

# Einführung in R

## Teil 1.2

Linda T. Betz, MSc  
Kambeitz Lab  
University Hospital Cologne  
April 8, 2020



**KAMBEITZ LAB**

Prediction and Prevention in  
Mental Health

# Zusammenfassung vom letzten Mal

- Daten umformatieren mit dem R package *tidyverse*

“Verben” für die Transformation von Daten:

- Auswahl von Beobachtungen anhand ihres Wertes mit `filter()`.
- Umordnen von Beobachtungen mit `arrange()`.
- Auswahl von Variablen mit `select()`.
- Erzeugen neuer Variablen anhand existierender Variablen mit `mutate()`.

NA als Sonderfall - viele Operationen auf Daten, in denen NA vorkommt ergeben NA

# Besprechung der Übungen

Übung A

in R

# Besprechung der Übungen

Warum ergibt `NA ^ 0` nicht `NA`?

=> `NA` ist hier ein Platzhalter für eine Zahl unbekannten Wertes. Jede Zahl hoch 0 ergibt 1. Deswegen ist es egal, welchen Wert `NA` hat, das Ergebnis ist immer 1 (und nicht `NA`).

Warum ist `NA | TRUE` nicht `NA`?

=> Der Operator “`|`” bedeutet “*oder*” und prüft, ob **mindestens eine von zwei** Bedingungen wahr (`TRUE`) ist. `NA` ist hier ein Platzhalter für einen logischen Ausdruck (`TRUE` oder `FALSE`). Egal, welchen Wert `NA` einnimmt, das Ergebnis ist immer `TRUE` (weil der andere Teil der Bedingungen auf jeden Fall `TRUE` ist).

# Besprechung der Übungen

```
filter(flights, dest == IAH)
```

Error: object 'IAH' not found

=> Name von Elementen in Daten (hier “IAH” als Kürzel für den Flughafen Houston, gespeichert in der Variable `dest`) muss immer mit Anführungszeichen aufgerufen werden. Sonst interpretiert R `IAH` als Objekt (das hier nicht existiert und deswegen einen Fehler ausgibt).

```
filter(flights, dest == “IAH”)
```

# Besprechung der Übungen

```
mutate(flights, distance_km = distance * 1,60934)
```

=> ergibt erst mal so keinen Fehler. ABER wenn wir die neue Dataframe betrachten, sehen wir, dass da was schiefgelaufen ist:

```
mutate(flights,
```

```
distance_km = distance * 1.60934)
```

=> IMMER Dezimalpunkt verwenden!

flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour	distance_km	60934
1545	N14228	EWR	IAH	227	1400	5	15	2013-01-01 05:00:00	1400	60934
1714	N24211	LGA	IAH	227	1416	5	29	2013-01-01 05:00:00	1416	60934
1141	N619AA	JFK	MIA	160	1089	5	40	2013-01-01 05:00:00	1089	60934
725	N804JB	JFK	BQN	183	1576	5	45	2013-01-01 05:00:00	1576	60934
461	N668DN	LGA	ATL	116	762	6	0	2013-01-01 06:00:00	762	60934
1696	N39463	EWR	ORD	150	719	5	58	2013-01-01 05:00:00	719	60934
507	N516JB	EWR	FLL	158	1065	6	0	2013-01-01 06:00:00	1065	60934
5708	N829AS	LGA	IAD	53	229	6	0	2013-01-01 06:00:00	229	60934
79	N593JB	JFK	MCO	140	944	6	0	2013-01-01 06:00:00	944	60934
301	N3ALAA	LGA	ORD	138	733	6	0	2013-01-01 06:00:00	733	60934
49	N793JB	JFK	PBI	149	1028	6	0	2013-01-01 06:00:00	1028	60934
71	N657JB	JFK	TPA	158	1005	6	0	2013-01-01 06:00:00	1005	60934
194	N29129	JFK	LAX	345	2475	6	0	2013-01-01 06:00:00	2475	60934
1124	N53441	EWR	SFO	361	2565	6	0	2013-01-01 06:00:00	2565	60934
707	N3701AA	LGA	DFW	257	1389	6	0	2013-01-01 06:00:00	1389	60934



# TEIL III: Zusammenfassen von Variablen



# Zusammenfassen von Variablen

`summarise()`

```
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))  
  
#> # A tibble: 1 x 1  
  
#>   delay  
  
#>   <dbl>  
  
#> 1  12.6
```

Was passiert hier?

# Zusammenfassen von Variablen

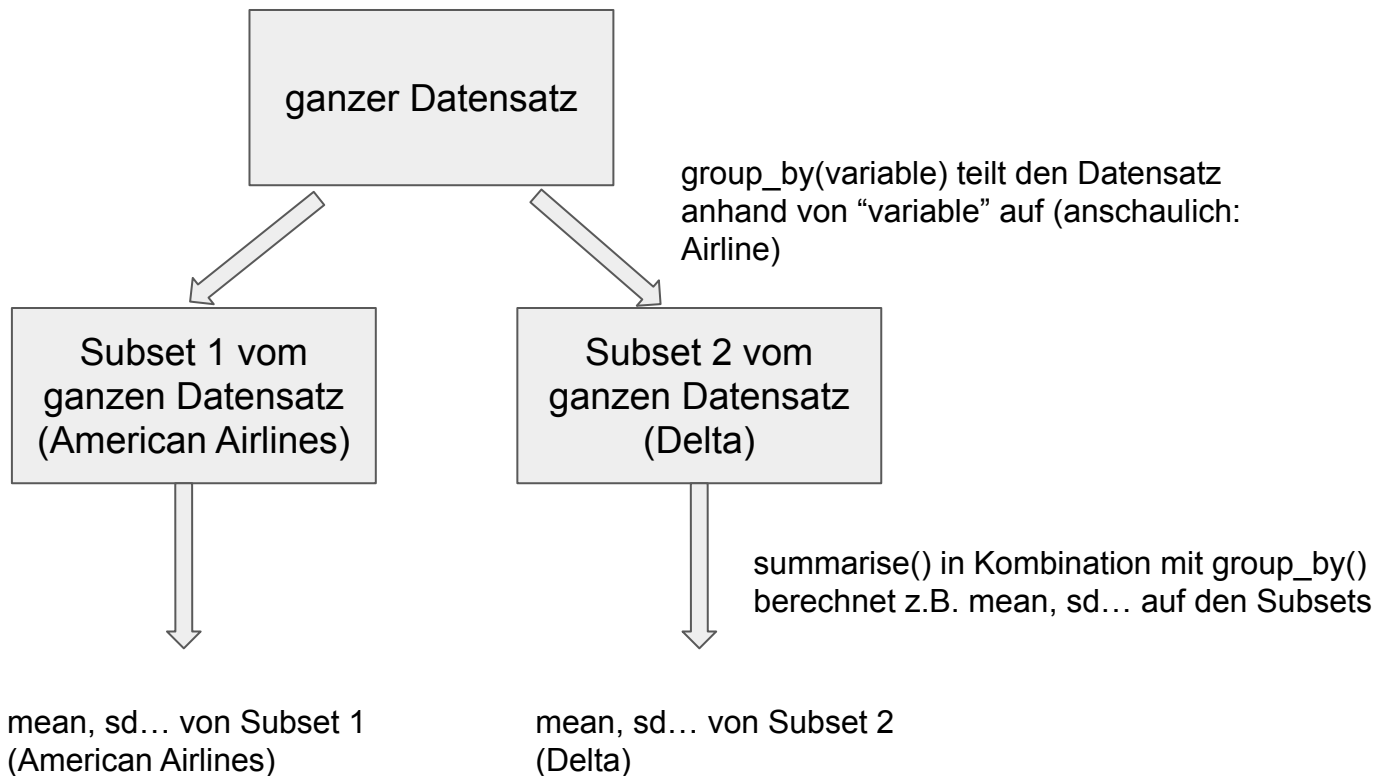
Gruppierte Zusammenfassung von Variablen

Was passiert hier?

```
by_day <- group_by(flights, year, month, day)
summarise(by_day, delay = mean(dep_delay, na.rm = TRUE))

#> # A tibble: 365 x 4
#> # Groups:   year, month [12]
#>   year month   day delay
#>   <int> <int> <int> <dbl>
#> 1  2013     1     1  11.5
#> 2  2013     1     2  13.9
#> 3  2013     1     3  11.0
#> 4  2013     1     4   8.95
#> 5  2013     1     5   5.73
#> 6  2013     1     6   7.15
#> # ... with 359 more rows
```

# Zusammenfassen von Variablen



# Zusammenfassen von Variablen

```
by_day <- group_by(flights, year, month, day)
summarise(by_day, delay = mean(dep_delay, na.rm = TRUE))
```

Das Argument `na.rm=TRUE` führt dazu, dass fehlende Werte vor den Berechnungen entfernt werden.

Entfernen wir es nicht, ergeben sich Ergebnisse als `NA`.

```
#> # Groups:   year, month [12]
#>   year month   day mean
#>   <int> <int> <int> <dbl>
#> 1  2013     1     1    NA
#> 2  2013     1     2    NA
#> 3  2013     1     3    NA
#> 4  2013     1     4    NA
#> 5  2013     1     5    NA
#> 6  2013     1     6    NA
```

# Übersichtliche Datentransformation

```
flights_sat <- filter(flights,  
  dest == "SAT")
```

```
arrange(flights_sat,  
  desc(arr_delay))
```

```
flights %>%
```

```
  filter(dest == "SAT") %>%
```

```
  arrange(desc(arr_delay))
```

Mittels des “Pipe-Operators” (%>%) können wir verschiedene Transformationen einfach zusammenschreiben (ohne, dass wir immer neue Objekte abspeichern müssen). Die transformierten Daten werden immer in den nächsten Schritt “weitergegeben”. Es wird intuitiv von oben nach unten gelesen.

## ***Cognitive process:***

1. Take the **ydat** dataset, *then*
2. **filter()** for genes in the leucine biosynthesis pathway, *then*
3. **group\_by()** the limiting nutrient, *then*
4. **summarize()** to correlate rate and expression, *then*
5. **mutate()** to round *r* to two digits, *then*
6. **arrange()** by rounded correlation coefficients

## ***The old way:***

```
arrange(  
  mutate(  
    summarize(  
      group_by(  
        filter(ydat, bp=="leucine biosynthesis"),  
        nutrient),  
      r=cor(rate, expression)),  
    r=round(r, 2)),  
  r)
```

## ***The dplyr way:***

= tidyverse way

```
ydat %>%  
  filter(bp=="leucine biosynthesis") %>%  
  group_by(nutrient) %>%  
  summarize(r=cor(rate, expression)) %>%  
  mutate(r=round(r,2)) %>%  
  arrange(r)
```

# Ein weiteres Beispiel

```
flights_jan <- filter(flights, month  
== 1)
```

```
flights_jan_airline <-  
group_by(flights_jan, carrier)
```

```
flights_jan_airline_delay <-  
summarize(flights_jan_airline,  
mean_delay = mean(dep_delay, na.rm =  
TRUE))
```

Was passiert hier?

Wir bestimmen für den Monat Januar pro Airline die mittlere Verspätung bei Abflug.

```
flights %>%
```

```
filter(month == 1) %>%
```

```
group_by(carrier) %>%
```

```
summarize(mean_delay = mean(dep_delay,  
na.rm = TRUE))
```



Funktionen, die wir in  
“summarize” verwenden  
können, um Variablen  
zusammenzufassen

Objective	Function	Description
Basic	mean()	Average of vector x
	median()	Median of vector x
	sum()	Sum of vector x
variation	sd()	standard deviation of vector x
	IQR()	Interquartile of vector x
Range	min()	Minimum of vector x
	max()	Maximum of vector x
	quantile()	Quantile of vector x
Position	first()	Use with group_by() First observation of the group
	last()	Use with group_by(). Last observation of the group
	nth()	Use with group_by(). nth observation of the group
Count	n()	Use with group_by(). Count the number of rows
	n_distinct()	Use with group_by(). Count the number of distinct observations

# Zusammenfassen von Variablen

```
flights %>%
```

```
filter(month == 1) %>%
```

```
group_by(carrier) %>%
```

```
summarize(mean_delay = mean(dep_delay, na.rm = TRUE),
```

```
          sd_delay = sd(dep_delay, na.rm = TRUE))
```

# Zusammenfassen von mehreren Variablen

```
flights %>%
```

```
summarize_at(vars(c(dep_delay, arr_delay)), mean, na.rm = TRUE)
```

```
flights %>%
```

summarize\_at() erlaubt direkten Zugriff auf Variablen mit  
Helferfunktionen, ähnlich zu select(), muss aber mit  
vars() quotiert werden

```
summarize_at(vars(contains("delay")), mean, na.rm = TRUE)
```

```
flights %>%
```

Hier werden mit logischer Operation alle  
Variablen ausgewählt, die numerics (also integer  
oder double) sind

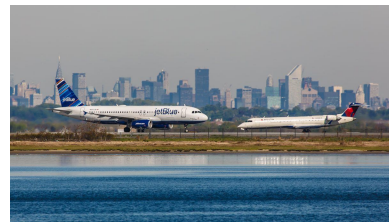
```
summarize_if(is.numeric, mean, na.rm = TRUE)
```

```
flights %>%
```

Mehrere Funktionen (mean, sd...) können als Vektor (mit c() ) eingefügt werden

```
summarize_if(is.numeric, c(mean, sd), na.rm = TRUE)
```

# Übungen (Übung D) - mit %>%



1. Erstelle eine neue Variable, die die Distanz zum Zielflughafen in Kilometern (statt Meilen) angibt.
2. Was war die Verspätung (bei Ankunft) im Mittel pro Flug für jede Airline?
3. Was war im Mittel die Verspätung (bei Ankunft) von Flügen von New York nach San Antonio, Texas?
4. Wie viele Flüge hatte die Airline Delta (DL) in jedem Monat des Jahres 2013 (Funktion `n()`)?
5. Finde für alle Variablen, die Doubles sind, den kleinsten Wert (Funktion `min()`).
6. Welche waren die 3 am meisten angeflogenen Flughäfen?
7. BONUS: Finde zu jeder Airline die größte Verspätung bei Ankunft (Funktion `max()`), die es 2013 gab. Welche Airline ist Spitzenreiter?
8. BONUS: Welche waren die 5 Flughäfen, wo es im Mittel bei Ankunft die größten Verspätungen bei Ankunft gab?
9. BONUS: Was war im Mittel die Verspätung bei Ankunft von Flügen von New York zu den 3 am meisten angeflogenen Zielflughäfen?

# Geschafft!



# Outlook

Das war nur der Anfang! R und das *tidyverse* erlaubt vieles, vieles mehr...

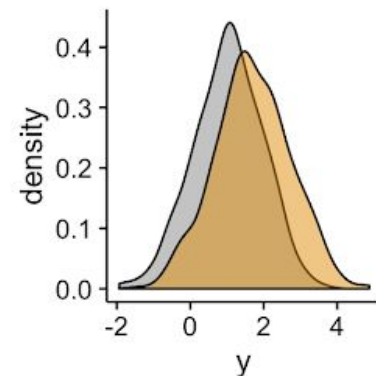
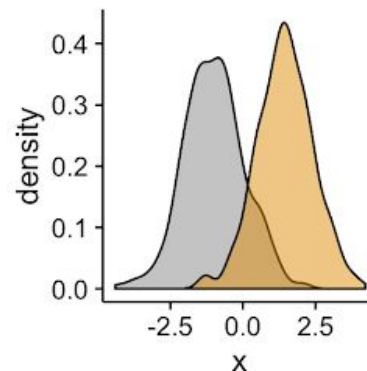
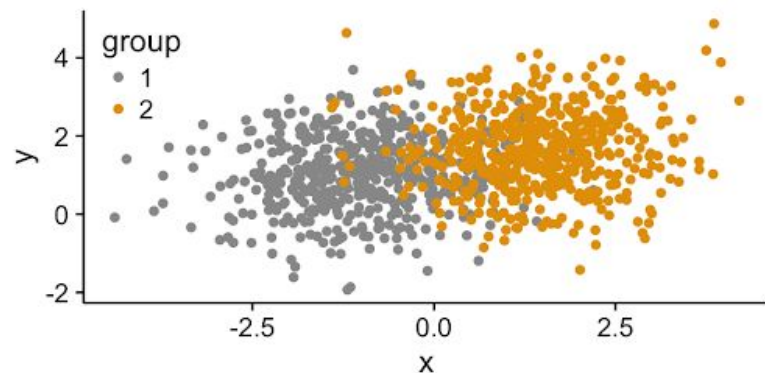
“Advanced” Data Wrangling

Tolle Plots

Komplexe Analysen (Netzwerke, Machine Learning...)

...

Einen Teil davon nächstes Mal mit Julian!



# Teil 2 des Kurses

... in 1 Woche (Mittwoch, 15. April) von 15-17 Uhr.

Weitere Infos (Installation von Packages, etc.) folgen.