

Datenanalyse und -visualisierung

mit der Programmiersprache R

Datenanalyse und -visualisierung

mit der Programmiersprache R

Pekka Sagner M.Sc.

-  sagner@iwkoeln.de
-  iwkoeln.de/.../pekka-sagner
-  [@pekkasagner](https://twitter.com/pekkasagner)

Wintersemester 2021/22

Letzte Aktualisierung: 02. Dezember 2021

Daten verbinden

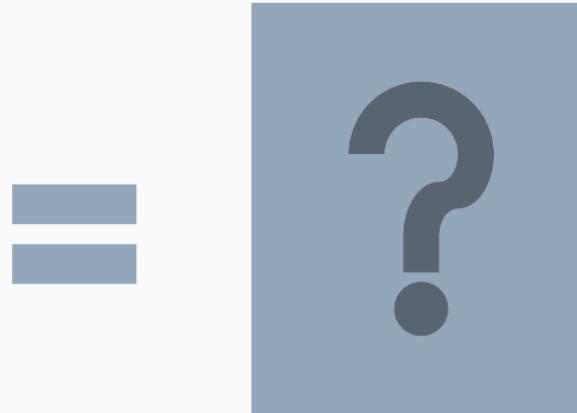
Inhalte und Ziele der Sitzung

- Kennenlernen der verschiedenen Arten, Datensätze zusammenzuführen

Datensatz a	
x1	x2
A	1
B	2
C	3



Datensatz b	
x1	x3
A	T
B	F
D	T



- Leseempfehlungen:
 - R4DS, Kapitel 13 (Wickham/Grolemund, 2021)

»Ein Datensatz kommt selten allein.«

Relationale Daten

- Es kommt eher selten vor, dass eine Datenanalyse nur auf einen einzigen Datensatz zurückgreift.
- Oft kombinieren wir in einem Analyseprojekt Daten aus verschiedenen Quellen, um unsere Forschungsfragen zu beantworten und so spannende Sachverhalte aufzudecken.
- Zusammen werden diese Datensätze als **relationale Daten** bezeichnet.
- Es ist die Beziehung zwischen den Daten, die wichtig ist, nicht die individuellen Datensätze alleine.
- Die Beziehung von Datensätzen ist immer zwischen **zwei** Datensätzen definiert.

Relationale Daten

- Für die Arbeit mit relationalen Daten brauchen wir Funktionen, die mit Datensatzpaaren funktionieren.
- Grundsätzlich gibt es drei Familien von Funktionen, um mit relationalen Daten zu arbeiten:
 - **Mutating Joins:** fügen einem Datensatz Variablen hinzu, basierend auf identischen Beobachtungen.
 - **Filtering Joins:** filtern Beobachtungen eines Datensatzes, basierend darauf, ob diese in einem anderen Datensatz vorkommen oder nicht.
 - **Set Operations:** behandeln die Beobachtungen wie mathematische Mengen und vergleichen die Werte jeder Variable.
- Die häufigsten Funktionen, die wir nutzen sind aus der Familie der **Mutating Joins**.

Die Familie der **Mutating Joins**

- **Mutating Joins** kommen immer dann zum Einsatz, wenn Informationen aus verschiedenen Datensätzen in einem neuen Datensatz zusammengefasst werden.
- `mutate()` ⇔ **Mutating Join**
- In der Regel ist dabei unser Ziel, Informationen aus verschiedenen Datensätzen zu kombinieren und somit neue Erkenntnisse zu gewinnen.

Key-Konzept: Keys

- Datensätze werden immer anhand einer oder mehrerer gemeinsamer Identifikationsvariablen (**Keys**, häufig auch **ID**) zusammengefügt.
- Welche Variablen sich dafür konkret anbieten, hängt vom Datensatz ab.
- Ein einfaches Beispiel:

X		Y	
1	x1	1	y1
2	x2	2	y2
3	x3	4	y3

Key-Konzept: Keys

- In R übersetzt sehen die Beispieldatensätze wie unten dargestellt aus.
- Die Variable `key` ist in dem Fall die Variable, die die Beobachtungen eindeutig definiert, zum Beispiel der Name einer Person, die Nummer eines Landkreises,
- Anhand dieser Variable werden die Datensätze zusammengeführt.

```
(x <- tribble(  
  ~key, ~val_x,  
  1, "x1",  
  2, "x2",  
  3, "x3"  
) )
```

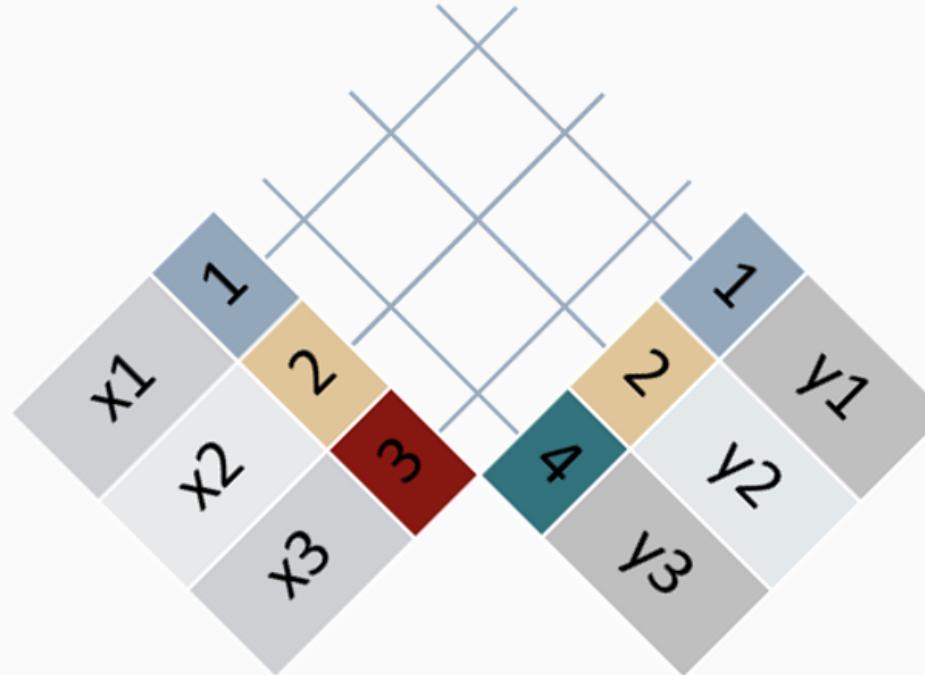
```
#> # A tibble: 3 x 2  
#>   key    val_x  
#>   <dbl> <chr>  
#> 1     1 x1  
#> 2     2 x2  
#> 3     3 x3
```

```
(y <- tribble(  
  ~key, ~val_y,  
  1, "y1",  
  2, "y2",  
  4, "y3"  
) )
```

```
#> # A tibble: 3 x 2  
#>   key    val_y  
#>   <dbl> <chr>  
#> 1     1 y1  
#> 2     2 y2  
#> 3     4 y3
```

Ziel der Mutating Joins

- Ein **Join** ist letztlich ein Weg, die Zeilen (Beobachtungen) in x mit **keiner, einer, oder mehreren** Zeilen (Beobachtungen) in y zusammenzuführen.
- Grafisch lassen sich alle möglichen Matches wie folgt darstellen:



- Jeder Schnittpunkt zweier Linien zeigt im Diagramm ein mögliches Match der Beobachtungen.
- Tatsächliche Matches werden mit Punkten markiert.

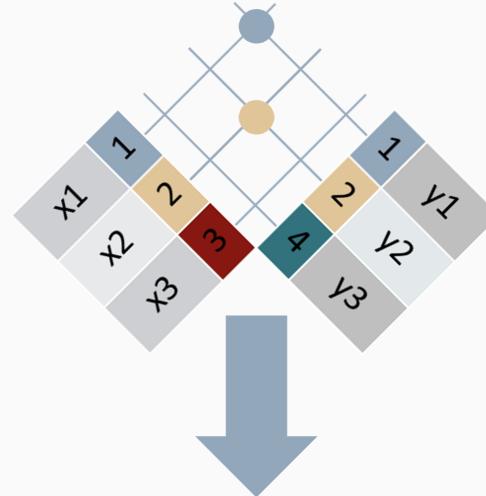
- Der einfachste Join ist ein **Inner Join**.
- Ein **Inner Join** fügt Paare von Beobachtungen zusammen, wenn die `key`-Variable identisch ist.
- Der Output der Funktion `inner_join()` ist ein Datensatz, der die `key`-Variable sowie die Werte-Variablen enthält.
- Das Argument `by` steht für die Variable(n), die die Identifikationsvariable(n) (`key`-Variable) bezeichnet.

Ein **Inner Join** ist dadurch charakterisiert, dass der zusammengeführte Datensatz nur Beobachtungen enthält, für die es ein Match gab.

Alle anderen ungematchten Beobachtungen gehen verloren.

Inner Join

Grafisch:



key	val_x	val_y
1	x1	y1
2	x2	y2

in R:

```
x %>%
  inner_join(y, by = "key")
```

```
#> # A tibble: 2 x 3
#>   key    val_x val_y
#>   <dbl> <chr>  <chr>
#> 1     1     x1     y1
#> 2     2     x2     y2
```

- Das Gegenstück zum Inner Join sind **Outer Joins**.

Inner Join

Ergebnis enthält Beobachtungen, die in beiden Datensätzen auftreten.

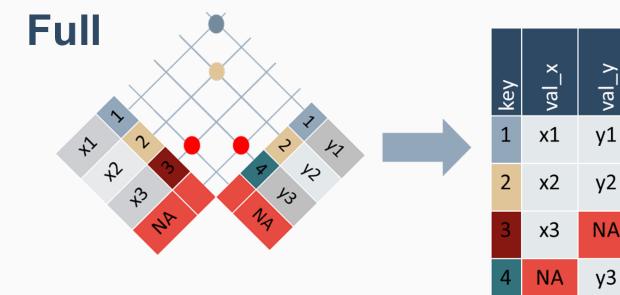
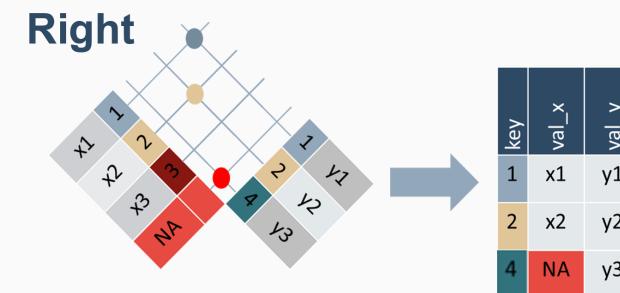
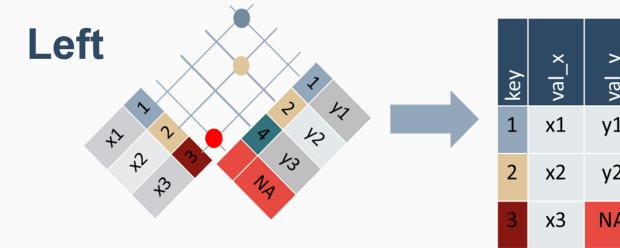
Outer Joins

Ergebnis enthält Beobachtungen, die **in mindestens einem** der beiden Datensätzen auftreten.

Funktionsweise der Mutating Joins

Outer Joins

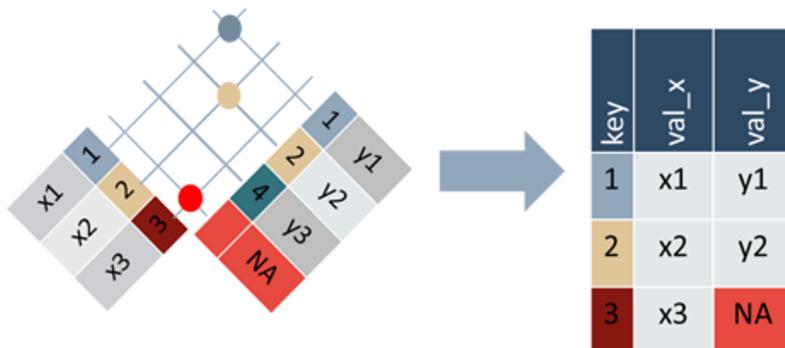
- Es gibt also nur eine Variante eines Inner Joins, aber **drei** Outer-Joins:
 - Left Join
 - Right Join
 - Full Join



Left Join

- Das Ergebnis eines **Left Join** enthält alle Beobachtungen des **linken**/ersten Datensatzes.
- Der **linke**/erste Datensatz wird um Variablen aus dem rechten/zweiten Datensatz ergänzt.

Grafisch:



in R:

```
x %>%
  left_join(y, by = "key")
```

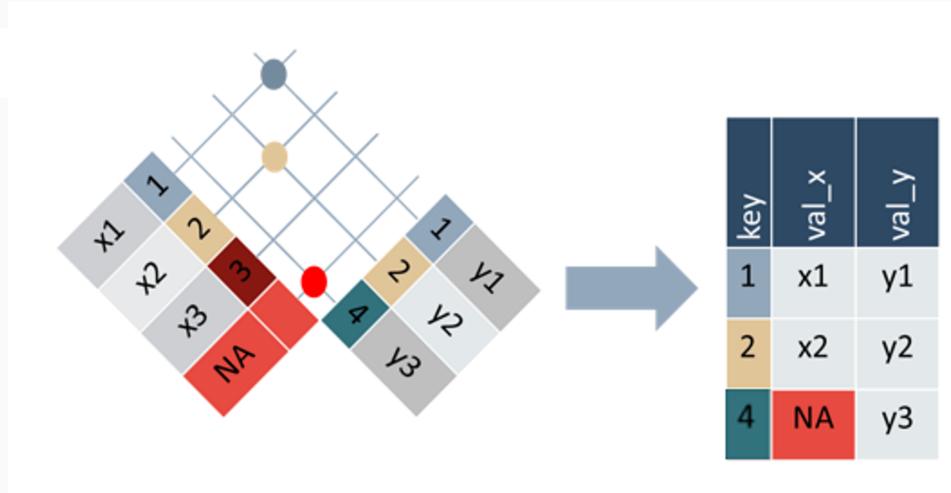
```
#> # A tibble: 3 x 3
#>   key    val_x  val_y
#>   <dbl> <chr>   <chr>
#> 1     1     x1     y1
#> 2     2     x2     y2
#> 3     3     x3     <NA>
```

Right Join

- Das Ergebnis eines **Right Join** enthält alle Beobachtungen des **rechten**/zweiten Datensatzes.
- Der **rechte**/zweite Datensatz wird um Variablen aus dem linken/ersten Datensatz ergänzt.

Grafisch:

in R:



```
x %>%
  right_join(y, by = "key")
```

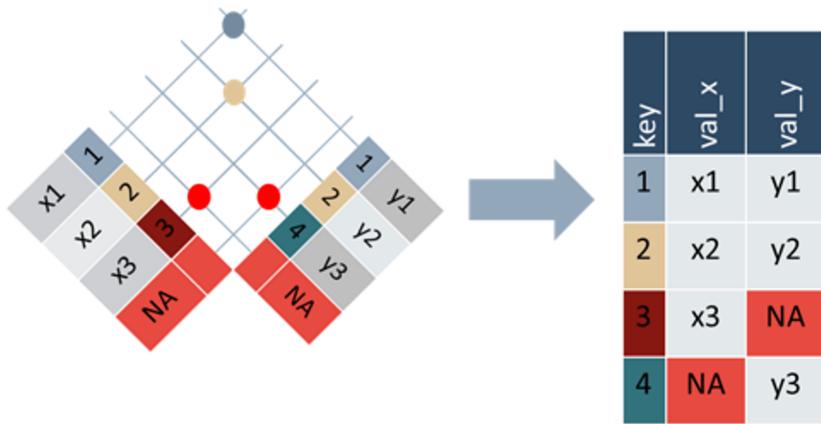
```
#> # A tibble: 3 x 3
#>   key    val_x  val_y
#>   <dbl> <chr>   <chr>
#> 1     1     x1     y1
#> 2     2     x2     y2
#> 3     4     <NA>   y3
```

Full Join

- Das Ergebnis eines **Full Join** enthält alle Beobachtungen **beider** Datensätze.
- Keine Beobachtungen gehen verloren.

Grafisch:

in R:



```
x %>%  
  full_join(y, by = "key")
```

```
#> # A tibble: 4 x 3  
#>   key    val_x  val_y  
#>   <dbl> <chr>  <chr>  
#> 1     1    x1    y1  
#> 2     2    x2    y2  
#> 3     3    x3    <NA>  
#> 4     4    <NA>  y3
```

- Der meistgenutzte Join ist der **Left Join**.
- Dieser kommt immer dann zum Einsatz, wenn ein bestehender Datensatz mit weiteren Informationen ergänzt werden soll.
- Dabei gehen keine Informationen aus dem Ausgangsdatensatz verloren.
- Das ist ein guter Standard.
- Das heißt, die Entscheidung für andere Joins sollte immer eine bewusste sein, die auf guten Gründen basiert.

Im Zweifel: Left Join

Mehrere, identische Keys

- In der Praxis kommt es häufiger vor, dass ein Datensatz Beobachtungen enthält, die durch eine Key -Variable bestimmt sind, die nicht einzigartig ist.
- Dies kommt zum Beispiel dann vor, wenn eine Beobachtung einer Einheit und einer Untereinheit zugewiesen werden kann.
- Zum Beispiel:

Landkreis → Bundesland → Land → Kontinent

Art → Gattung → Familie → Ordnung

Wichtige Erweiterungen

Mehrere, identische Keys

```
(x <- tribble(  
  ~key, ~val_x,  
  1, "x1",  
  2, "x2",  
  2, "x3",  
  1, "x4"  
) )
```

```
#> # A tibble: 4 x 2  
#>   key    val_x  
#>   <dbl> <chr>  
#> 1     1 x1  
#> 2     2 x2  
#> 3     2 x3  
#> 4     1 x4
```

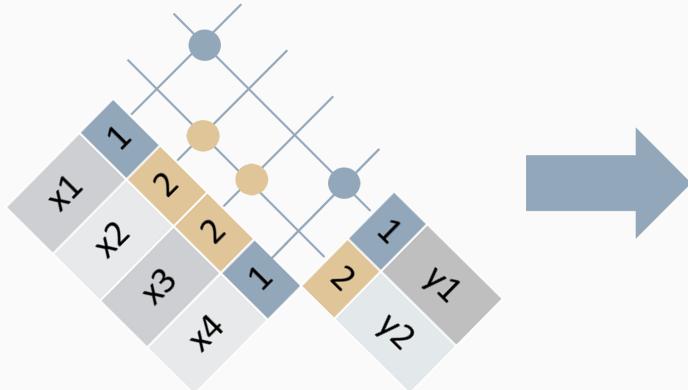
```
(y <- tribble(  
  ~key, ~val_y,  
  1, "y1",  
  2, "y2"  
) )
```

```
#> # A tibble: 2 x 2  
#>   key    val_y  
#>   <dbl> <chr>  
#> 1     1 y1  
#> 2     2 y2
```

Wichtige Erweiterungen

Mehrere, identische Keys

Grafisch:



key	val_x	val_y
1	x1	y1
2	x2	y2
2	x3	y2
1	x4	y1

in R:

```
x %>%
  left_join(y, by = "key")
```

```
#> # A tibble: 4 x 3
#>   key  val_x val_y
#>   <dbl> <chr> <chr>
#> 1     1    x1    y1
#> 2     2    x2    y2
#> 3     2    x3    y2
#> 4     1    x4    y1
```

Wichtige Erweiterungen

Keys mit unterschiedlichen Namen

- Bisher entsprachen sich die Namen der identifizierenden `key`-Variable in den beiden Datensätzen.
- Falls dies einmal nicht der Fall sein sollte, so können die Namen der `key`-Variablen, die sich inhaltlich entsprechen, händisch definiert werden:

```
(x <- tribble(  
  ~key_x, ~val_x,  
  1, "x1",  
  2, "x2",  
  3, "x3"  
))
```

```
#> # A tibble: 3 x 2  
#>   key_x val_x  
#>   <dbl> <chr>  
#> 1     1 x1  
#> 2     2 x2  
#> 3     3 x3
```

```
(y <- tribble(  
  ~key_y, ~val_y,  
  1, "y1",  
  2, "y2",  
  4, "y3"  
))
```

```
#> # A tibble: 3 x 2  
#>   key_y val_y  
#>   <dbl> <chr>  
#> 1     1 y1  
#> 2     2 y2  
#> 3     4 y3
```

Wichtige Erweiterungen

Keys mit unterschiedlichen Namen

```
x %>%  
  left_join(y, by = c("key_x" = "key_y"))
```

```
#> # A tibble: 3 x 3  
#>   key_x val_x val_y  
#>   <dbl> <chr> <chr>  
#> 1     1   x1    y1  
#> 2     2   x2    y2  
#> 3     3   x3    <NA>
```

- Die Reihenfolge beim Gleichsetzen der Key-Variablen entspricht dabei der Reihenfolge der Datensätze.
- Der vergebene Key-Variablenname im neuen Datensatz ist der erste.

Genug der Theorie. Ab nach

