

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

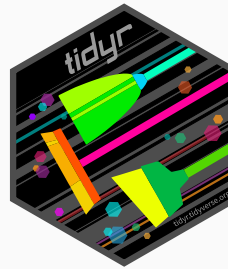
🐙 [@pekkasagner](https://github.com/pekkasagner)

Wintersemester 2022-23

Säubern von Daten

Inhalte und Ziele der Sitzung

- Umgang mit »unsauberen« Daten.
- Transformieren von Daten vom Wide-Format ins Long-Format und vice-versa.
- Aufteilen und Zusammenführen von Spalten.



Quelle: [github/rstudio.com](https://github.com/rstudio.com)

- Leseempfehlungen:
 - R4DS, Kapitel 12 (Wickham/Grolemund, 2021)
 - Data Organization in Spreadsheets (Broman/Woo, 2018)
 - Tidy Data (Wickham, 2014)

Tidy Data?

(Wiederholung)

Tidy Data?

(Wiederholung)

country	year	cases	population
Afghanistan	1999	18215	1752071
Afghanistan	2000	18666	20095360
Brazil	1999	31737	17206362
Brazil	2000	84488	17404898
China	1999	212258	127205272
China	2000	212258	128005583

Variablen

Jede Variable in einer eigenen Spalte.

country	year	cases	population
Afghanistan	1999	18215	1752071
Afghanistan	2000	18666	20095360
Brazil	1999	31737	17206362
Brazil	2000	84488	17404898
China	1999	212258	127205272
China	2000	212258	128005583

Beobachtungen

Jede Beobachtung in einer eigenen Zeile.

country	year	cases	population
Afghanistan	1999	18215	1752071
Afghanistan	2000	18666	20095360
Brazil	1999	31737	17206362
Brazil	2000	84488	17404898
China	1999	212258	127205272
China	2000	212258	128005583

Werte

Jeder Wert in einer eigenen Zelle.

Messy Data \rightsquigarrow Tidy Data

Messy Data \rightsquigarrow Tidy Data

“Tidy datasets are all alike, but every messy dataset is messy in its own way.” ~ Hadley Wickham

- Der erste Schritt, um unordentliche Daten in ein ordentliches Format zu überführen, ist es, die Datenstruktur zu verstehen.
- Danach muss geprüft werden, ob die drei Bedingungen für ordentliche Daten erfüllt sind.
- Steht jede Variable in einer eigenen Spalte?
Nein? \rightarrow
- Steht jede Beobachtung in einer eigenen Zeile?
Nein? \rightarrow
- Steht jeder Wert in einer eigenen Zelle?
Nein? \rightarrow



Quelle: [github/rstudio.com](https://github.com/rstudio.com)

- Für die Transformation von Spalten und Zeilen, **das Drehen** eines Datensatzes, bietet das Paket `tidyr` aus dem Tidyverse die Funktionen

`pivot_longer()` und `pivot_wider()`.

Transformieren von Spalten

Zwei Darstellungsformen derselben Daten

wide				long		
id	x	y	z	id	key	val
1	a	c	e	1	x	a
2	b	d	f	2	x	b
				1	y	c
				2	y	d
				1	z	e
				2	z	f

Quelle: Garrick Aiden-Buie

Transformieren von Spalten ins Long-Format

Beispiel **Wide-to-Long**:

land_a	land_b	land_c
1000000	1100000	1230000

Ziel:

land	bip
a	1000000
b	1100000
c	1230000

pivot_longer()

- Die Funktion `pivot_longer()` macht einen Datensatz **länger**.
- Man sagt auch, man bringt einen Datensatz vom **Wide-Format** ins **Long-Format**.
- Ein einfaches, häufig auftretendes Beispiel für **Wide-Formate** sind Tabellen im Zusammenhang mit Jahreszahlen:

	2018	2019	2020	2021
bip	123	231	321	113

- Diese Darstellung ist zwar gut menschenlesbar, aber schlecht maschinenlesbar.
- Wide-Formate sind (meistens) nicht Tidy.

Unsere Beispieldaten (wide-to-long):

```
(messy_data_1 <- tibble(land_a = 1000000,  
                        land_b = 1100000,  
                        land_c = 1230000)  
)
```

```
#> # A tibble: 1 x 3  
#>   land_a land_b land_c  
#>   <dbl> <dbl> <dbl>  
#> 1 1000000 1100000 1230000
```

pivot_longer()

- Im einfachsten Fall braucht die Funktion nur einen Input.
- `cols` bezeichnet die Spalten, die in das Long-Format übertragen werden sollen.

```
messy_data_1 %>%  
  pivot_longer(cols = everything())
```

```
#> # A tibble: 3 x 2  
#>   name      value  
#>   <chr>    <dbl>  
#> 1 land_a 1000000  
#> 2 land_b 1100000  
#> 3 land_c 1230000
```

- Alternativ zur Hilfsfunktion `everything()`, die uns einfach alle Spalten auswählen lässt, könnten wir die Spalten auch explizit als Character-Vektor definieren.

pivot_longer()

- Mit `names_to` wird der Name für die Spalte vergeben, deren Inhalt aus den **Spaltennamen** des alten Datensatzes besteht.

```
messy_data_1 %>%  
  pivot_longer(cols = everything(),  
               names_to = "land")
```

```
#> # A tibble: 3 x 2  
#>   land      value  
#>   <chr>    <dbl>  
#> 1 land_a 1000000  
#> 2 land_b 1100000  
#> 3 land_c 1230000
```

pivot_longer()

- Mit `values_to` wird der Name für die Spalte vergeben, deren Inhalt aus den **Zellwerten** des alten Datensatzes besteht.

```
messy_data_1 %>%  
  pivot_longer(cols = everything(),  
               names_to = "land",  
               values_to = "bip")
```

```
#> # A tibble: 3 x 2  
#>   land      bip  
#>   <chr>    <dbl>  
#> 1 land_a 1000000  
#> 2 land_b 1100000  
#> 3 land_c 1230000
```

pivot_longer()

- Mit dem `names_prefix`-Argument können wir schließlich das nicht benötigte `land_`-Präfix entfernen.

```
messy_data_1 %>%  
  pivot_longer(cols = everything(),  
               names_to = "land",  
               values_to = "bip",  
               names_prefix = "land_")
```

```
#> # A tibble: 3 x 2  
#>   land      bip  
#>   <chr>   <dbl>  
#> 1 a      1000000  
#> 2 b      1100000  
#> 3 c      1230000
```


`pivot_wider()`

- Die Funktion `pivot_wider()` macht einen Datensatz **breiter**.
- Man sagt auch, man bringt einen Datensatz vom **Long-Format** *ins* **Wide-Format**.
- **Tidy Data** ist im **Long-Format**.
- Aber Daten im **Long-Format** sind nicht zwingend **Tidy**.
- Deshalb kann es durchaus auch im Sinne des Tidy-Data-Prinzips sinnvoll und notwendig sein, Datensätze in ein breiteres Format zu bringen.

Transformieren von Spalten ins Wide-Format

Beispiel **Long-to-Wide**:

land	typ	wert
a	bip	1000000
a	einwohner	5000
b	bip	1100000
b	einwohner	6000
c	bip	1230000
c	einwohner	4000

Ziel:

land	bip	einwohner
a	1000000	5000
b	1100000	6000
c	1230000	4000

Unsere Beispieldaten (Long-to-Wide):

```
(messy_data_2 <- tribble(~ land, ~ typ, ~ wert,  
  "a", "bip", 1000000,  
  "a", "einwohner", 5000,  
  "b", "bip", 1100000,  
  "b", "einwohner", 6000,  
  "c", "bip", 1230000,  
  "c", "einwohner", 4000)  
)
```

```
#> # A tibble: 6 x 3  
#>   land typ      wert  
#>   <chr> <chr>    <dbl>  
#> 1 a     bip      1000000  
#> 2 a     einwohner  5000  
#> 3 b     bip      1100000  
#> 4 b     einwohner  6000  
#> 5 c     bip      1230000  
#> 6 c     einwohner  4000
```

- Hinweis: `tribble()` erlaubt die zeilenweise Definition von Tibbles.

pivot_wider()

- In diesem simplen Fall braucht die Funktion mindestens zwei Inputs, die als Paar **zusammenspielen**.
- `names_from` bezeichnet die Spalte(n), aus der die Werte für die Zielspaltennamen stammen.
- `values_from` bezeichnet die Spalte(n) aus denen die Werte für die entsprechenden Zielspalten stammen.

```
messy_data_2 %>%  
  pivot_wider(names_from = typ,  
              values_from = wert)
```

```
#> # A tibble: 3 x 3  
#>   land      bip einwohner  
#>   <chr>   <dbl>     <dbl>  
#> 1 a      1000000     5000  
#> 2 b      1100000     6000  
#> 3 c      1230000     4000
```

`pivot_wider()` und `pivot_wider()`

- Die beiden Funktionen bieten noch zahlreiche Argumente, die in besonderen Konstellationen hilfreich sein können.
- Beispiele für `pivot_longer()` sind hier beschrieben: https://tidyr.tidyverse.org/reference/pivot_longer.html
- Beispiele für `pivot_wider()` sind hier beschrieben: https://tidyr.tidyverse.org/reference/pivot_wider.html

Ein weiteres Beispiel für unsaubere Daten

Beispiel **Teilen von Spalten:**

land	bip/einwohner
a	1000000/5000
b	1100000/6000

Ziel:

land	bip	einwohner
a	1000000	5000
b	1100000	6000

Unsere Beispieldaten (separate):

```
(messy_data_3 <- tribble(~ land, ~ "bip/einwohner",  
  "a", "1000000/5000",  
  "b", "1100000/6000")  
)
```

```
#> # A tibble: 2 x 2  
#>   land `bip/einwohner`  
#>   <chr> <chr>  
#> 1 a      1000000/5000  
#> 2 b      1100000/6000
```


separate()

- `separate` erhält als erstes Argument `col` die Spalte, in der wir die Zellenwerte trennen möchten.
- `into` definiert die Spaltennamen der neuen getrennten Spalten.
- `sep` bezeichnet den Trenn-Character.

```
messy_data_3 %>%  
  separate(col = "bip/einwohner",  
           into = c("bip", "einwohner"),  
           sep = "/")
```

```
#> # A tibble: 2 x 3  
#>   land bip    einwohner  
#>   <chr> <chr>   <chr>  
#> 1 a    1000000 5000  
#> 2 b    1100000 6000
```

- Der Gegenpart von `separate()` ist die Funktion `unite()`.

Genug der Theorie. Ab nach  Studio®.