

# Datenanalyse und -visualisierung

## mit der Programmiersprache R

---

# Datenanalyse und -visualisierung

## mit der Programmiersprache R

Pekka Sagner M.Sc.

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

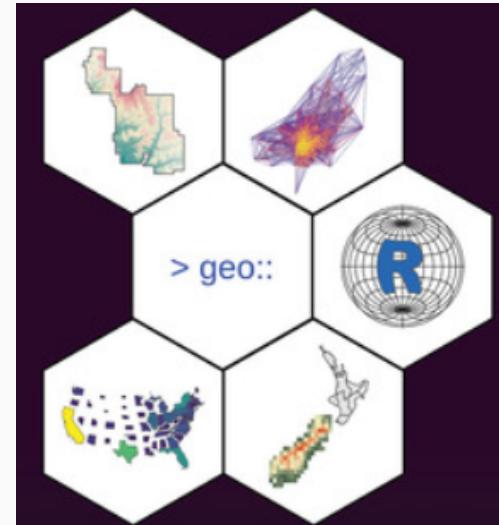
Sommersemester 2022

## Geodaten

Letzte Aktualisierung: 08. Juni 2022

# Inhalte und Ziele der Sitzung

- Kennenlernen der Grundstruktur von Geodaten
- Visualisierung von Geodaten

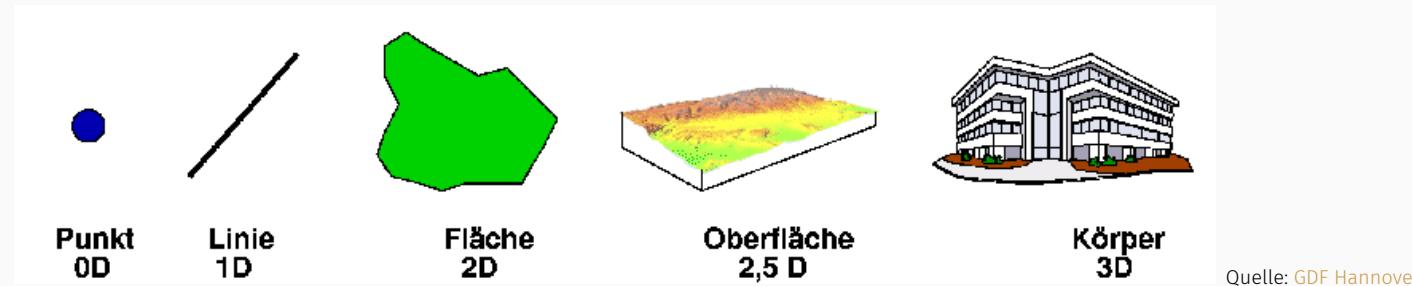


- Leseempfehlungen:
  - Geocomputation with R (Lovelace et al., 2021)
  - ggplot2: Elegant Graphics for Data Analysis, Kapitel 6.2 ff (Wickham et al., 2021)
  - Vignettes for the sf-package (Pebesma et al., 2021)

# Was sind Geodaten?

# Was sind Geodaten?

- Geodaten sind Informationen, die im Zusammenhang mit der Erde stehen. Sie beschreiben eine räumliche Lage.
- Geodaten können in ihre **geometrische Form** (engl: shape/geometry) und ihre zugehörigen **Sachinformationen (Attribute)** geteilt werden.
- Beispiele für Geodaten:



# Geodaten leicht gemacht

- Die Visualisierung von simplen Geoinformationen unterscheidet sich nicht wesentlich von der Visualisierung anderer Daten.
- Bei Punktdaten im zweidimensionalen Raum bedarf es beispielsweise eines x-Wertes und eines y-Wertes, um diesen zu visualisieren.
- Übertragen auf die Erdkugel bedarf es eines Längen- und Breitengrades.
- Hinzu kommt eine **Projektion** (z.B. Mercator-Projektion) und ein **geodätisches Datum**.
- Beide Informationen zusammen bilden ein **Koordinatenreferenzsystem**.
- Das geodätische Datum trifft Annahmen bezüglich des Erdmittelpunkts, des Ursprungs von Längen und Breitengrad, usw.
- Das Koordinatenreferenzsystem bestimmt, wie die elipsoide Erdoberfläche auf einen zweidimensionalen Raum (Karte) übertragen wird.
- Mehr Informationen in ggplot2: Elegant Graphics for Data Analysis, Kapitel 6.3 (Wickham et al., 2021).

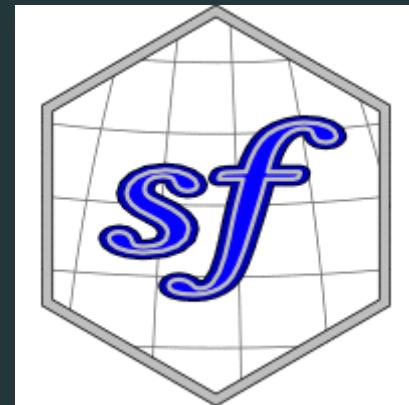
# Geodaten leicht gemacht

- Geodaten werden in vielerlei Formen bereitgestellt. Zu den häufig verwendeten gehört das Shapefile-Format von ESRI.
- Weitere geläufige Geodatenformate sind:
  - kml
  - GeoJSON
- R kann mit den meisten Geodatenformaten umgehen.

# Geodaten in den Sozialwissenschaften

- Die Visualisierung von Attributen im geografischen Raum bietet sich dann an, wenn wir die regionalen Unterschiede in den Attributen betonen möchten.
- Oft liegen uns sozio-ökonomische Daten auf der Ebene von Raumeinheiten vor.
- Eine Landkarte, die die Ausprägung eines Attributs in einer Raumeinheit darstellt, erweitert anderweitige Abbildungen um eine weitere Dimension.
- R kann noch viel mehr als das, z. B. die Berechnung von Distanzen, geografische Verschneidungen, 3D-Visualisierungen, ...

# Geodaten mit dem Paket `sf`



- Das Paket `sf` erlaubt uns das Arbeiten mit Geodaten, in der uns bekannten `Tidyverse`-Syntax.

## Arbeit mit Geodaten anhand eines Beispiels:

- Ziel:** Visualisierung der **durchschnittlichen Arbeitslosenquote** in den **Bundesländern**.
- Neben einigen Funktionen aus dem Paket `sf` brauchen wir, wie immer, Daten:
  - Im Beispiel sind das die Daten, die wir visualisieren möchten (**Attribute**), hier die Arbeitslosenquote je Bundesland,
  - und zusätzlich die **Geometrien**, hier die Umrisse der Bundesländer.

# Quellen für Geodaten

- Eine Google Suche á la: **Shapefile "Name der Raumeinheit"** oder **geojson "Name der Raumeinheit"** fördert meist das gewünschte Ergebnis zu Tage.
- Shapefiles für die deutschen Verwaltungsgebiete stellt zum Beispiel das **Bundesamt für Kartographie und Geodäsie** bereit.
- Es gibt darüber hinaus einige **R**-Pakete, die den Download von Geoinformationen direkt aus **R** heraus ermöglichen.
- Besonderheit bei **Shapefiles**:
  - Ein Shapefile ist keine einzelne Datei, es besteht aus mindestens drei Dateien:
  - **.shp** dient zur Speicherung der Geometriedaten
  - **.dbf** Sachdaten (auch Attributdaten genannt) im dBASE-Format
  - **.shx** dient als Index der Geometrie zur Verknüpfung der Sachdaten
  - Beim Einladen der Daten in **R** spielt das jedoch keine Rolle.

# Praxisbeispiel

Geoinformationen einlesen

# Geoinformationen einlesen

- Um ein Shapefile einzulesen, wählen wir eines der Shapefile-Teile aus.
- Wir lesen Geodaten mit der Funktion `read_sf()` aus dem Paket `sf` ein.
- Die Funktion kann mit vielen weiteren Geodatenformaten umgehen.

```
library(tidyverse)
library(sf)

bundeslaender <- read_sf("data/Bundesland_clean/Bundesland_shape_clean.shp")
```

- `sf` steht für **Simple Features**, eine »**allgemein gültige Architektur für geografische Daten**«.
- Für die Visualisierung relevant ist, dass sich `sf`-Objekte mit der bekannten `ggplot`-Syntax visualisieren lassen.
- `sf`-Objekte können darüber hinaus mit der bekannten Syntax verändert werden.

## Struktur eines `sf`-Objekts

bundeslaender

```
#> Simple feature collection with 16 features and 2 fields
#> Geometry type: MULTIPOLYGON
#> Dimension:      XY
#> Bounding box:  xmin: 280353.1 ymin: 5235878 xmax: 921261.6 ymax: 6101302
#> Projected CRS: ETRS89 / UTM zone 32N
#> # A tibble: 16 x 3
#>   AGS     GEN                                geometry
#>   <chr> <chr>                               <MULTIPOLYGON [m]>
#> 1 01 Schleswig-Holstein (((428802.8 6004127, 427486.6 6003257, 426225.6 6005019, 428337.4 6004906, 429478.9 ...
#> 2 02 Hamburg (((464521.5 5978903, 462755.7 5977541, 461955.5 5977811, 463123.4 5978641, 463129.9 ...
#> 3 03 Niedersachsen (((358703.9 5939438, 357875.2 5940047, 358627.9 5940777, 359379 5940364, 358703.9 59...
#> 4 04 Bremen (((471012 5933445, 468446.4 5938237, 467569.6 5939474, 468140.2 5939786, 470697.3 59...
#> 5 05 Nordrhein-Westfalen (((497895.6 5706077, 492410 5703432, 494508.2 5699391, 494405.4 5697709, 496185.7 56...
#> 6 06 Hessen (((437411.5 5569859, 437085.4 5572675, 436232.1 5573150, 436456.3 5574899, 434912.8 ...
#> 7 07 Rheinland-Pfalz (((375299.1 5490019, 375152.4 5491292, 374508.3 5492528, 374687.3 5493799, 373329 54...
#> 8 08 Baden-Württemberg (((477730.9 5284628, 478787.9 5282116, 478115.3 5281851, 476810.6 5282473, 474974.2 ...
#> 9 09 Bayern (((609304 5267896, 611412.8 5271124, 610698.9 5271411, 607945.5 5271010, 607563.7 52...
#> 10 10 Saarland (((361552.9 5446473, 360071.7 5445778, 360504 5443598, 359332.9 5443107, 358827 5441...
#> 11 11 Berlin (((791413.7 5842844, 790893.8 5842428, 791533.4 5841333, 791395.6 5839623, 794035.4 ...
#> 12 12 Brandenburg (((785946.5 5753693, 785639.8 5754990, 784317.3 5756006, 781094.8 5754818, 780962.7 ...
#> 13 13 Mecklenburg-Vorpommern (((781646 6013699, 782498.3 6012606, 784820.1 6013162, 785221.3 6012500, 782852.8 60...
#> 14 14 Sachsen (((753976.9 5653939, 753032 5655691, 751452.6 5654801, 751573.8 5653912, 749135.5 56...
#> 15 15 Sachsen-Anhalt (((668257.8 5690780, 667266.6 5690798, 666512.6 5693214, 666784.7 5695026, 665467.5 ...
#> 16 16 Thüringen (((726497.5 5648383, 727596.7 5650422, 727072.6 5650537, 727318.6 5651669, 730241.1 ...
```

- Die Datenstruktur eines `sf`-Objekts ist ein Tibble.
- Unterschiede zum klassischen Fall liegen in den Informationen hinsichtlich des Koordinatenreferenzsystems und der Variable `geometry`.

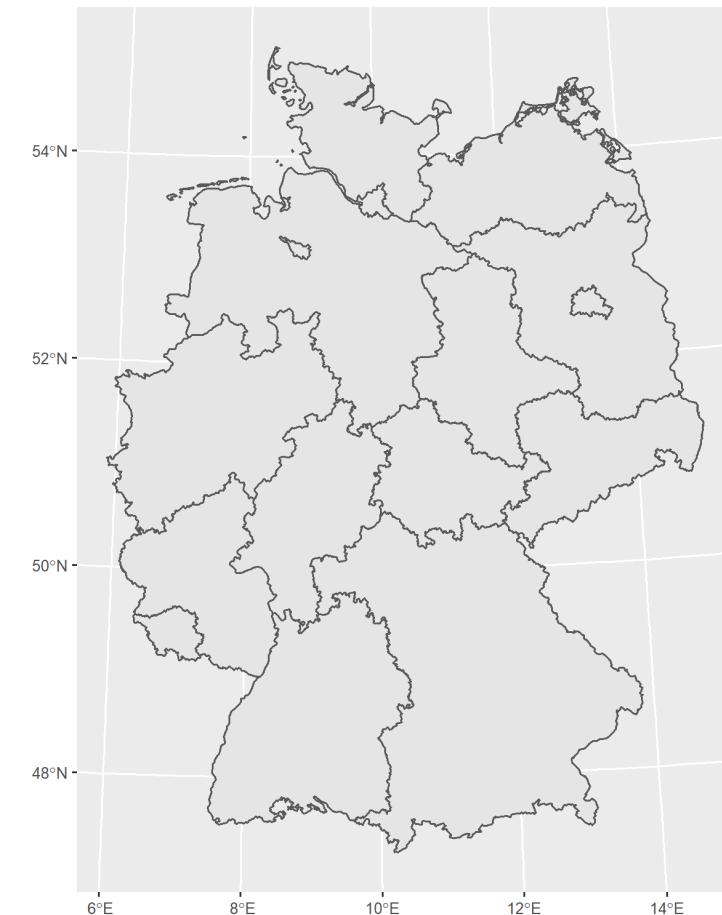
# Praxisbeispiel

## Rohe Geoinformationen visualisieren

# Rohe Geoinformationen visualisieren

```
bundeslaender %>%  
  ggplot(aes(geometry = geometry)) +  
  geom_sf()
```

- Bei der Visualisierung von `sf`-Objekten gibt es nur zwei wesentliche Unterschiede zu den uns bekannten Visualierungsformen.
- Anstelle einer `x` und/oder `y`-Aesthetic nutzen wir die ausschließlich `sf`-Objekten vorbehaltene `geometry`-Aesthetic. (Hier wird der `geometry`-Aesthetic die Variable mit dem Namen `geometry` zugewiesen.)
- Darüber hinaus nutzen wir das `geom`, das `sf`-Objekten vorbehalten ist: `geom_sf()`



# Praxisbeispiel

Geoinformationen und Attribute kombinieren

# Geoinformationen und Attribute kombinieren

- Bis hierhin haben wir nur die Geometrie selbst visualisiert.
- In der Regel ist das nicht das Ziel.
- Wir sind vielmehr am Attribut interessiert, dass wir mithilfe der Geometrie visualisieren möchten.
- Geometrien sind »**sticky**«.
- Das heißt, wir können wie gewohnt zwei Tabellen (Tibbles) miteinander verbinden.
- Wie? → Im Zweifel mit einem **Left Join**:

```
geodaten_mit_attributen <- rohgeodaten %>%
  left_join(attribute)
```

- Dabei steht auf der linken Seite des Joins das `sf`-Objekt, rechts der Tibble, der die Attribute enthält, die wir an die Geoinformationen anknüpfen.

# Attribute einladen

```
(alq <- read_csv("data/Arbeitslosenquote_Bundeslaender_clean.csv")
)
```

```
#> # A tibble: 16 x 3
#>   Kennziffer Raumeinheit      Arbeitslosenquote
#>   <chr>     <chr>            <dbl>
#> 1 01       Schleswig-Holstein    6
#> 2 02       Hamburg             6.8
#> 3 03       Niedersachsen      5.8
#> 4 04       Bremen              10.2
#> 5 05       Nordrhein-Westfalen 7.4
#> 6 06       Hessen              5
#> 7 07       Rheinland-Pfalz    4.8
#> 8 08       Baden-Württemberg  3.5
#> 9 09       Bayern              3.2
#> 10 10      Saarland            6.7
#> 11 11      Berlin               9
#> 12 12      Brandenburg        7
#> 13 13      Mecklenburg-Vorpommern 8.6
#> 14 14      Sachsen             6.7
#> 15 15      Sachsen-Anhalt     8.4
#> 16 16      Thüringen           6.1
```

# Geometrien und Attribute kombinieren

```
(joined_bundeslaender_alq <- bundeslaender %>%
    left_join(alq, by = c("AGS" = "Kennziffer")) # Auswahl der keys
)
```

```
#> Simple feature collection with 16 features and 4 fields
#> Geometry type: MULTIPOLYGON
#> Dimension:      XY
#> Bounding box:  xmin: 280353.1 ymin: 5235878 xmax: 921261.6 ymax: 6101302
#> Projected CRS: ETRS89 / UTM zone 32N
#> # A tibble: 16 x 5
#>   AGS     GEN                                geometry Raumeinheit Arbeitslosenquo~
#>   <chr> <chr>                                <MULTIPOLYGON [m]> <chr>          <dbl>
#> 1 01 Schleswig-Holstein (((428802.8 6004127, 427486.6 6003257, 426225.6 6005019, ~ Schleswig-Holste~ 6
#> 2 02 Hamburg (((464521.5 5978903, 462755.7 5977541, 461955.5 5977811, ~ Hamburg 6.8
#> 3 03 Niedersachsen (((358703.9 5939438, 357875.2 5940047, 358627.9 5940777, ~ Niedersachsen 5.8
#> 4 04 Bremen (((471012 5933445, 468446.4 5938237, 467569.6 5939474, 46~ Bremen 10.2
#> 5 05 Nordrhein-Westfalen (((497895.6 5706077, 492410 5703432, 494508.2 5699391, 49~ Nordrhein-Westfa~ 7.4
#> 6 06 Hessen (((437411.5 5569859, 437085.4 5572675, 436232.1 5573150, ~ Hessen 5
#> 7 07 Rheinland-Pfalz (((375299.1 5490019, 375152.4 5491292, 374508.3 5492528, ~ Rheinland-Pfalz 4.8
#> 8 08 Baden-Württemberg (((477730.9 5284628, 478787.9 5282116, 478115.3 5281851, ~ Baden-Württemberg 3.5
#> 9 09 Bayern (((609304 5267896, 611412.8 5271124, 610698.9 5271411, 60~ Bayern 3.2
#> 10 10 Saarland (((361552.9 5446473, 360071.7 5445778, 360504 5443598, 35~ Saarland 6.7
#> 11 11 Berlin (((791413.7 5842844, 790893.8 5842428, 791533.4 5841333, ~ Berlin 9
#> 12 12 Brandenburg (((785946.5 5753693, 785639.8 5754990, 784317.3 5756006, ~ Brandenburg 7
#> 13 13 Mecklenburg-Vorpommern (((781646 6013699, 782498.3 6012606, 784820.1 6013162, 78~ Mecklenburg-Vorp~ 8.6
#> 14 14 Sachsen (((753976.9 5653939, 753032 5655691, 751452.6 5654801, 75~ Sachsen 6.7
#> 15 15 Sachsen-Anhalt (((668257.8 5690780, 667266.6 5690798, 666512.6 5693214, ~ Sachsen-Anhalt 8.4
#> 16 16 Thüringen (((726497.5 5648383, 727596.7 5650422, 727072.6 5650537, ~ Thüringen 6.1
```

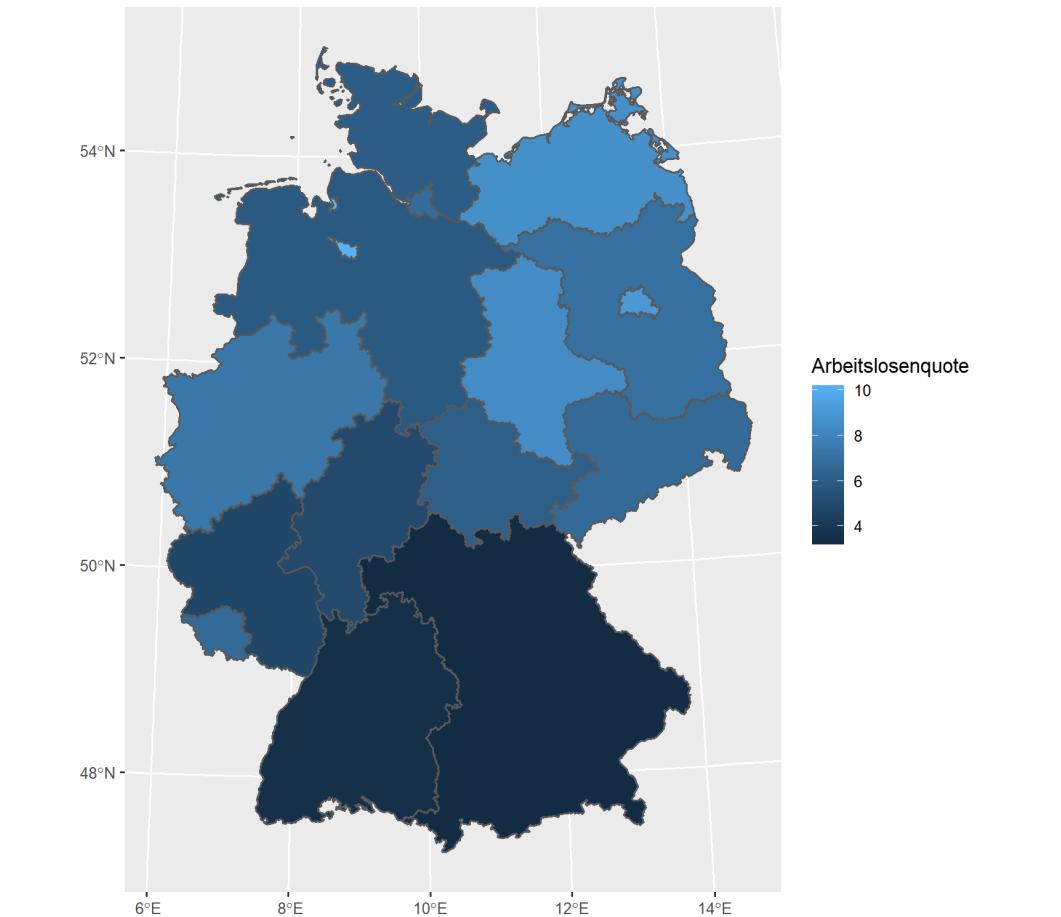
# Praxisbeispiel

Geometrien und Attribute visualisieren

# Geometrien und Attribute visualisieren

```
joined_bundeslaender_alq %>%  
  ggplot(aes(geometry = geometry,  
            fill = Arbeitslosenquote))  
  geom_sf()
```

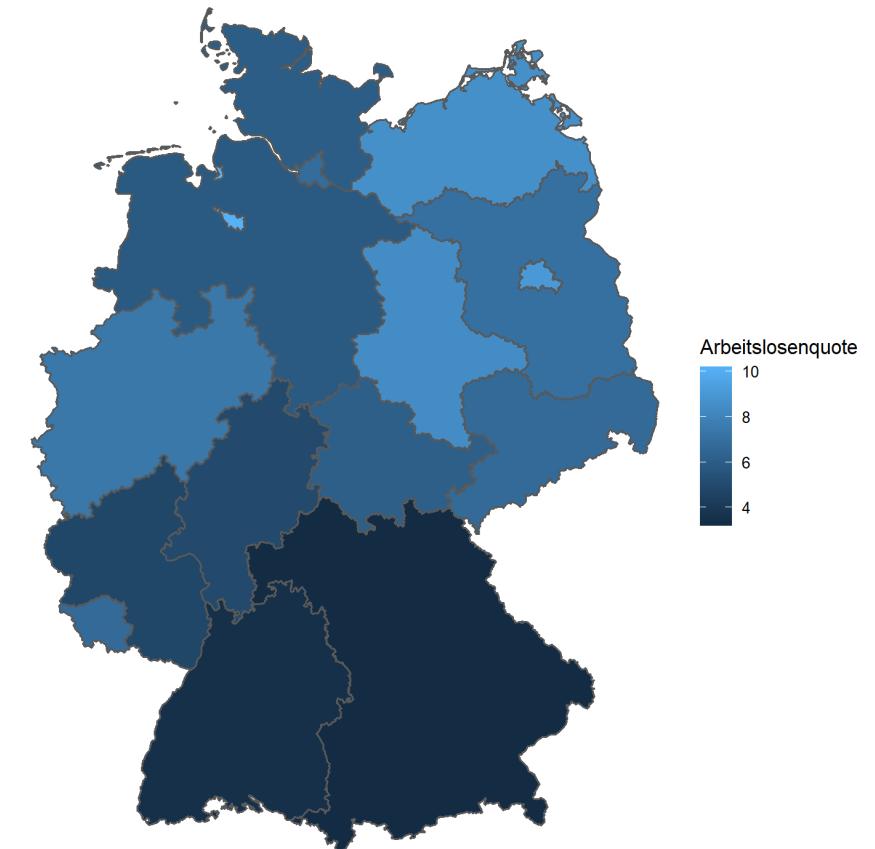
- Der `fill`-Aesthetic wird das zu visualisierende Attribut zugewiesen.
- Ist man an diesem Punkt angekommen, ist alles Weitere Kosmetik...



# Geometrien und Attribute visualisieren

```
joined_bundeslaender_alq %>%  
  ggplot(aes(geometry = geometry,  
             fill = Arbeitslosenquote))  
  geom_sf() +  
  theme_void()
```

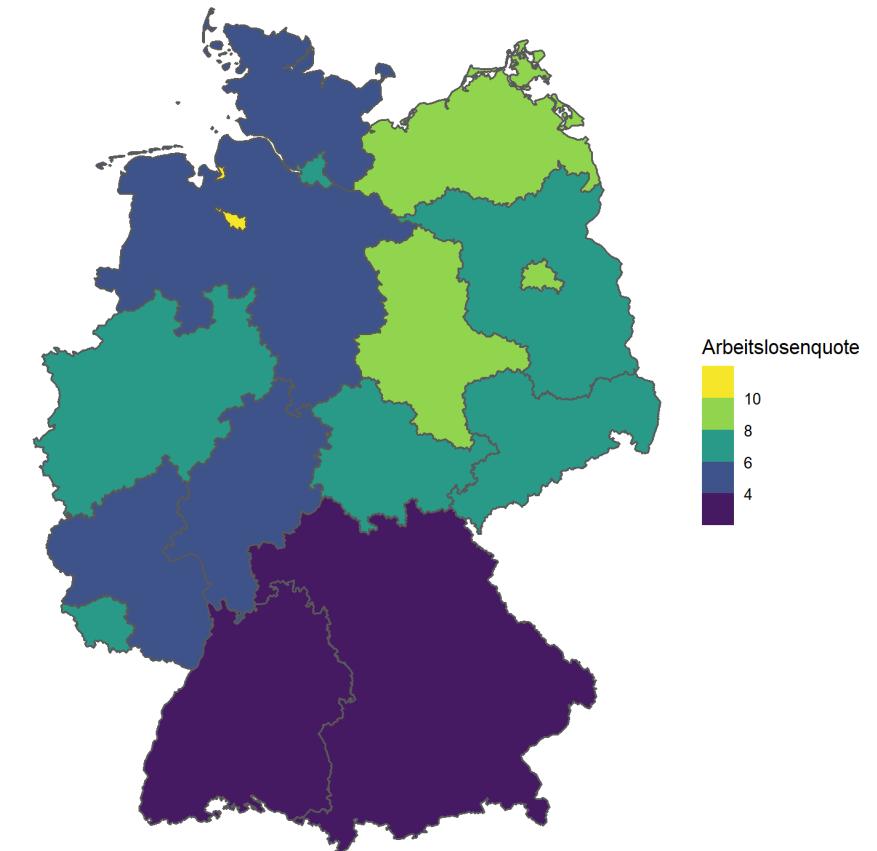
- So bietet es sich für Karten an, Längen- und Breitengrad sowie die entsprechenden Achsen im Hintergrund auszublenden.
- Hierfür kann man das Theme mit der Funktion `theme_void()` anpassen.



# Geometrien und Attribute visualisieren

```
joined_bundeslaender_alq %>%  
  ggplot(aes(geometry = geometry,  
             fill = Arbeitslosenquote))  
  geom_sf() +  
  theme_void() +  
  scale_fill_viridis_b()
```

- Die Farbskala lässt sich wie gewohnt anpassen.
- Optional: `scale_fill_viridis_b()` gruppiert die kontinuierliche Variable und weist dann erst die Farben zu.
- Die Gruppierung von Beobachtungen auf Basis von kontinuierlichen Variablen ist ein nützliches Tool.



# Exkurs

## Gruppieren von Daten

# Exkurs: Gruppieren von Daten

- Es gibt verschiedene Möglichkeiten, Daten auf Basis von kontinuierlichen Variablen zu Gruppieren.
- Eine bequeme Möglichkeit bieten die `cut_...`-Funktionen.
  - `cut_interval()` erstellt `n` Gruppen mit gleichem Intervall.
  - `cut_number()` erstellt `n` Gruppen mit (ungefähr) gleicher Anzahl an Beobachtungen.
  - `cut_width()` erstellt Gruppen der Breite `width`.
- Etwas flexibler ist die sehr nützliche Funktion `case_when()`.
  - Sie hat den entscheidenden Vorteil, dass sie auch für die Gruppierung von nichtnumerischen Variablen genutzt werden kann.
  - Logische Struktur der Funktion:

```
case_when(logischer Test 1 ~ mach das,  
          logischer Test 2 ~ mach dies,  
          logischer Test 3 ~ mach jenes,  
          ... ~ ...)
```

# Exkurs: Gruppieren von Daten

## cut\_interval()

```
alq %>%
  mutate(gruppe = cut_interval(Arbeitslosenquote, n = 4)) %>%
  add_count(gruppe) #wieviele Beob. je Gruppe?
```

```
#> # A tibble: 16 x 5
#>   Kennziffer Raumeinheit      Arbeitslosenquote gruppe     n
#>   <chr>      <chr>                <dbl> <fct>     <int>
#> 1 01        Schleswig-Holstein      6    (4.95,6.7]    4
#> 2 02        Hamburg                 6.8  (6.7,8.45]   6
#> 3 03        Niedersachsen          5.8  (4.95,6.7]    4
#> 4 04        Bremen                  10.2 (8.45,10.2]  3
#> 5 05        Nordrhein-Westfalen    7.4   (6.7,8.45]   6
#> 6 06        Hessen                  5    (4.95,6.7]    4
#> 7 07        Rheinland-Pfalz       4.8   [3.2,4.95]    3
#> 8 08        Baden-Württemberg     3.5   [3.2,4.95]    3
#> 9 09        Bayern                  3.2   [3.2,4.95]    3
#> 10 10       Saarland                6.7   (6.7,8.45]   6
#> 11 11       Berlin                  9    (8.45,10.2]  3
#> 12 12       Brandenburg            7    (6.7,8.45]   6
#> 13 13       Mecklenburg-Vorpommern 8.6   (8.45,10.2]  3
#> 14 14       Sachsen                 6.7   (6.7,8.45]   6
#> 15 15       Sachsen-Anhalt        8.4   (6.7,8.45]   6
#> 16 16       Thüringen               6.1   (4.95,6.7]   4
```

# Exkurs: Gruppieren von Daten

## cut\_number()

```
alq %>%
  mutate(gruppe = cut_number(Arbeitslosenquote, n = 4)) %>%
  add_count(gruppe) #wieviele Beob. je Gruppe?
```

```
#> # A tibble: 16 x 5
#>   Kennziffer Raumeinheit      Arbeitslosenquote gruppe     n
#>   <chr>      <chr>                <dbl> <fct>     <int>
#> 1 01        Schleswig-Holstein       6    (5.6,6.7]    5
#> 2 02        Hamburg                 6.8  (6.7,7.65]   3
#> 3 03        Niedersachsen          5.8  (5.6,6.7]    5
#> 4 04        Bremen                  10.2 (7.65,10.2]  4
#> 5 05        Nordrhein-Westfalen     7.4  (6.7,7.65]   3
#> 6 06        Hessen                  5    [3.2,5.6]    4
#> 7 07        Rheinland-Pfalz        4.8  [3.2,5.6]    4
#> 8 08        Baden-Württemberg      3.5  [3.2,5.6]    4
#> 9 09        Bayern                  3.2  [3.2,5.6]    4
#> 10 10       Saarland                6.7  (5.6,6.7]    5
#> 11 11       Berlin                  9    (7.65,10.2]  4
#> 12 12       Brandenburg            7    (6.7,7.65]   3
#> 13 13       Mecklenburg-Vorpommern  8.6  (7.65,10.2]  4
#> 14 14       Sachsen                 6.7  (5.6,6.7]    5
#> 15 15       Sachsen-Anhalt         8.4  (7.65,10.2]  4
#> 16 16       Thüringen              6.1  (5.6,6.7]    5
```

# Exkurs: Gruppieren von Daten

## cut\_width()

```
alq %>%
  mutate(gruppe = cut_width(Arbeitslosenquote, width = 2)) %>%
  add_count(gruppe) #wieviele Beob. je Gruppe?
```

```
#> # A tibble: 16 x 5
#>   Kennziffer Raumeinheit      Arbeitslosenquote gruppe     n
#>   <chr>      <chr>                <dbl> <fct>    <int>
#> 1 01        Schleswig-Holstein      6     (5,7]    7
#> 2 02        Hamburg                 6.8   (5,7]    7
#> 3 03        Niedersachsen          5.8   (5,7]    7
#> 4 04        Bremen                  10.2  (9,11]   1
#> 5 05        Nordrhein-Westfalen    7.4   (7,9]    4
#> 6 06        Hessen                  5     [3,5]    4
#> 7 07        Rheinland-Pfalz       4.8   [3,5]    4
#> 8 08        Baden-Württemberg     3.5   [3,5]    4
#> 9 09        Bayern                  3.2   [3,5]    4
#> 10 10       Saarland                6.7   (5,7]    7
#> 11 11       Berlin                  9     (7,9]    4
#> 12 12       Brandenburg            7     (5,7]    7
#> 13 13       Mecklenburg-Vorpommern 8.6   (7,9]    4
#> 14 14       Sachsen                 6.7   (5,7]    7
#> 15 15       Sachsen-Anhalt        8.4   (7,9]    4
#> 16 16       Thüringen               6.1   (5,7]    7
```

# Exkurs: Gruppieren von Daten

## case\_when()

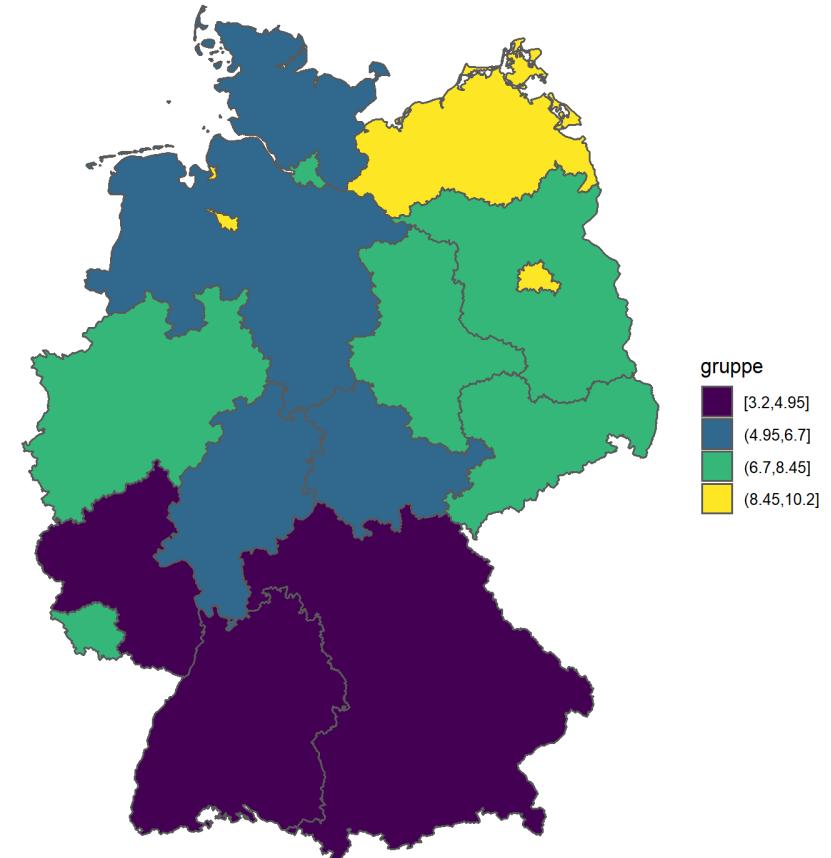
```
alq %>%
  mutate(gruppe = case_when(Arbeitslosenquote < 5 ~ "kleiner als 5 Prozent",
                            between(Arbeitslosenquote, 5, 8) ~ "5 bis 8 Prozent",
                            Arbeitslosenquote > 8 ~ "größer als 8 Prozent")) %>%
  add_count(gruppe) #wieviele Beob. je Gruppe?
```

```
#> # A tibble: 16 x 5
#>   Kennziffer Raumeinheit      Arbeitslosenquote gruppe     n
#>   <chr>      <chr>            <dbl> <chr>       <int>
#> 1 01        Schleswig-Holstein    6    5 bis 8 Prozent    9
#> 2 02        Hamburg             6.8  5 bis 8 Prozent    9
#> 3 03        Niedersachsen      5.8  5 bis 8 Prozent    9
#> 4 04        Bremen              10.2  größer als 8 Prozent  4
#> 5 05        Nordrhein-Westfalen  7.4  5 bis 8 Prozent    9
#> 6 06        Hessen              5    5 bis 8 Prozent    9
#> 7 07        Rheinland-Pfalz     4.8  kleiner als 5 Prozent  3
#> 8 08        Baden-Württemberg   3.5  kleiner als 5 Prozent  3
#> 9 09        Bayern              3.2  kleiner als 5 Prozent  3
#> 10 10       Saarland            6.7  5 bis 8 Prozent    9
#> 11 11       Berlin               9    größer als 8 Prozent  4
#> 12 12       Brandenburg         7    5 bis 8 Prozent    9
#> 13 13       Mecklenburg-Vorpommern 8.6  größer als 8 Prozent  4
#> 14 14       Sachsen              6.7  5 bis 8 Prozent    9
#> 15 15       Sachsen-Anhalt     8.4  größer als 8 Prozent  4
#> 16 16       Thüringen           6.1  5 bis 8 Prozent    9
```

# Exkurs: Gruppieren von Daten

## cut\_interval() - Visualisierung

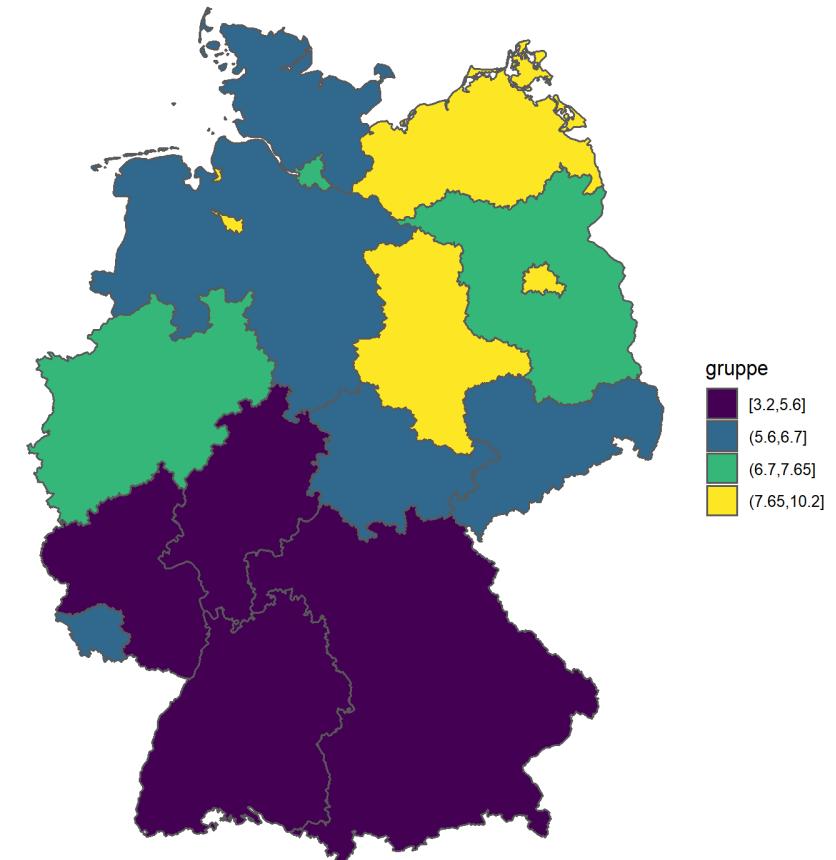
```
joined_bundeslaender_alq %>%
  mutate(gruppe = cut_interval(Arbeitslosenquote, n = 4)) %
  ggplot(aes(geometry = geometry,
             fill = gruppe)) +
  geom_sf() +
  theme_void() +
  scale_fill_viridis_d()
```



# Exkurs: Gruppieren von Daten

## cut\_number() - Visualisierung

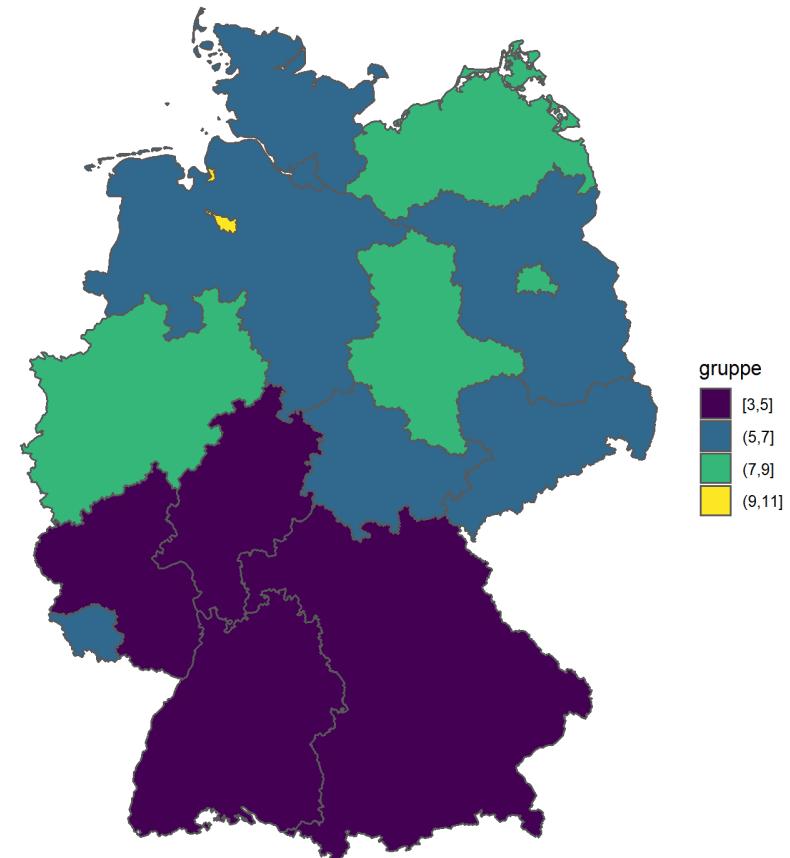
```
joined_bundeslaender_alq %>%  
  mutate(gruppe = cut_number(Arbeitslosenquote, n = 4)) %>%  
  ggplot(aes(geometry = geometry,  
             fill = gruppe)) +  
  geom_sf() +  
  theme_void() +  
  scale_fill_viridis_d()
```



# Exkurs: Gruppieren von Daten

## cut\_width() - Visualisierung

```
joined_bundeslaender_alq %>%  
  mutate(gruppe = cut_width(Arbeitslosenquote, width = 2.0))  
  ggplot(aes(geometry = geometry,  
             fill = gruppe)) +  
  geom_sf() +  
  theme_void() +  
  scale_fill_viridis_d()
```

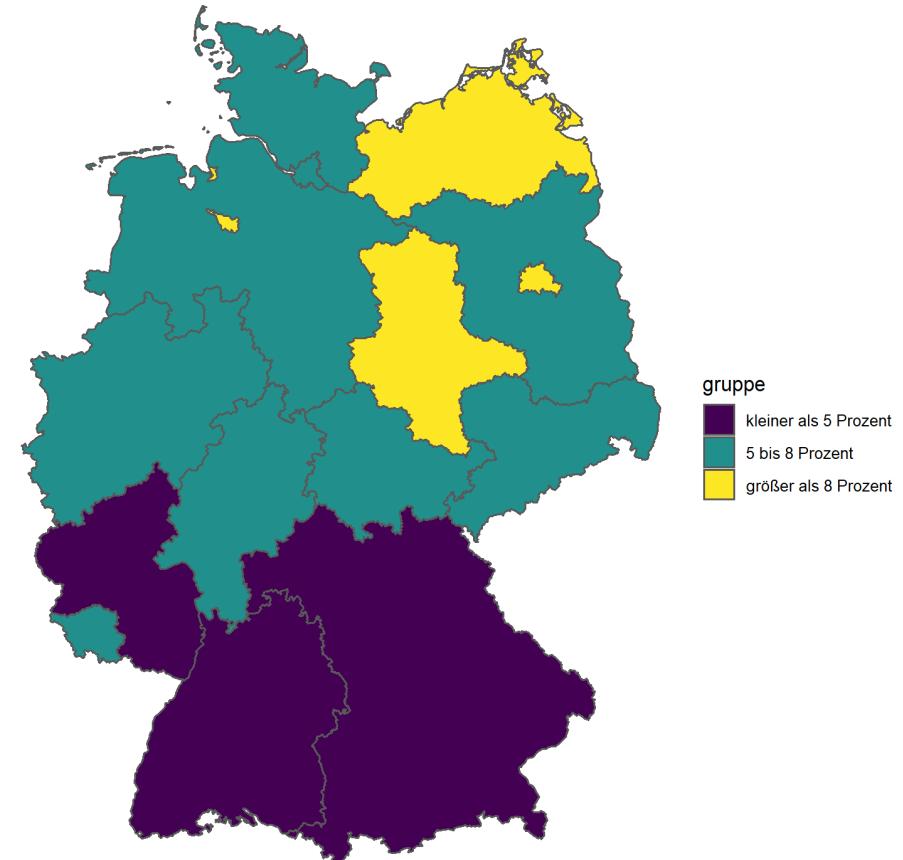


# Exkurs: Gruppieren von Daten

## case\_when() - Visualisierung

```
joined_bundeslaender_alq %>%  
  mutate(gruppe = case_when(  
    Arbeitslosenquote < 5 ~ "kleiner als 5 Prozent",  
    between(Arbeitslosenquote, 5, 8) ~ "5 bis 8 Prozent"  
    Arbeitslosenquote > 8 ~ "größer als 8 Prozent")) %>  
  mutate(gruppe = factor(gruppe,  
    levels = c("kleiner als 5 Prozent",  
              "5 bis 8 Prozent",  
              "größer als 8 Prozent"))) %>  
  ggplot(aes(geometry = geometry,  
             fill = gruppe)) +  
  geom_sf() +  
  theme_void() +  
  scale_fill_viridis_d()
```

- Im Gegensatz zu den `cut_...()`-Funktionen, erstellt `case_when()` keinen (geordneten) Faktorvektor, sondern einen Charaktervektor.
- Für die gewünschte Ordnung, bei der Visualisierung, erstellen wir also zuerst einen Faktorvektor und ordnen die `levels`.



Genug der Theorie. Ab nach

