Datenaufbereitung und -visualisierung

Termin 6

Jakob Kapeller

University of Duisburg-Essen
Institute for Socio-Economics &

Johannes Kepler University Linz
Institute for Comprehensive Analysis of the Economy (ICAE)

Editor: *Heterodox Economics Newsletter*

www.jakob-kapeller.org | www.uni-due.de | www.heterodoxnews.com

UNIVERSITÄT DUISBURG ESSEN

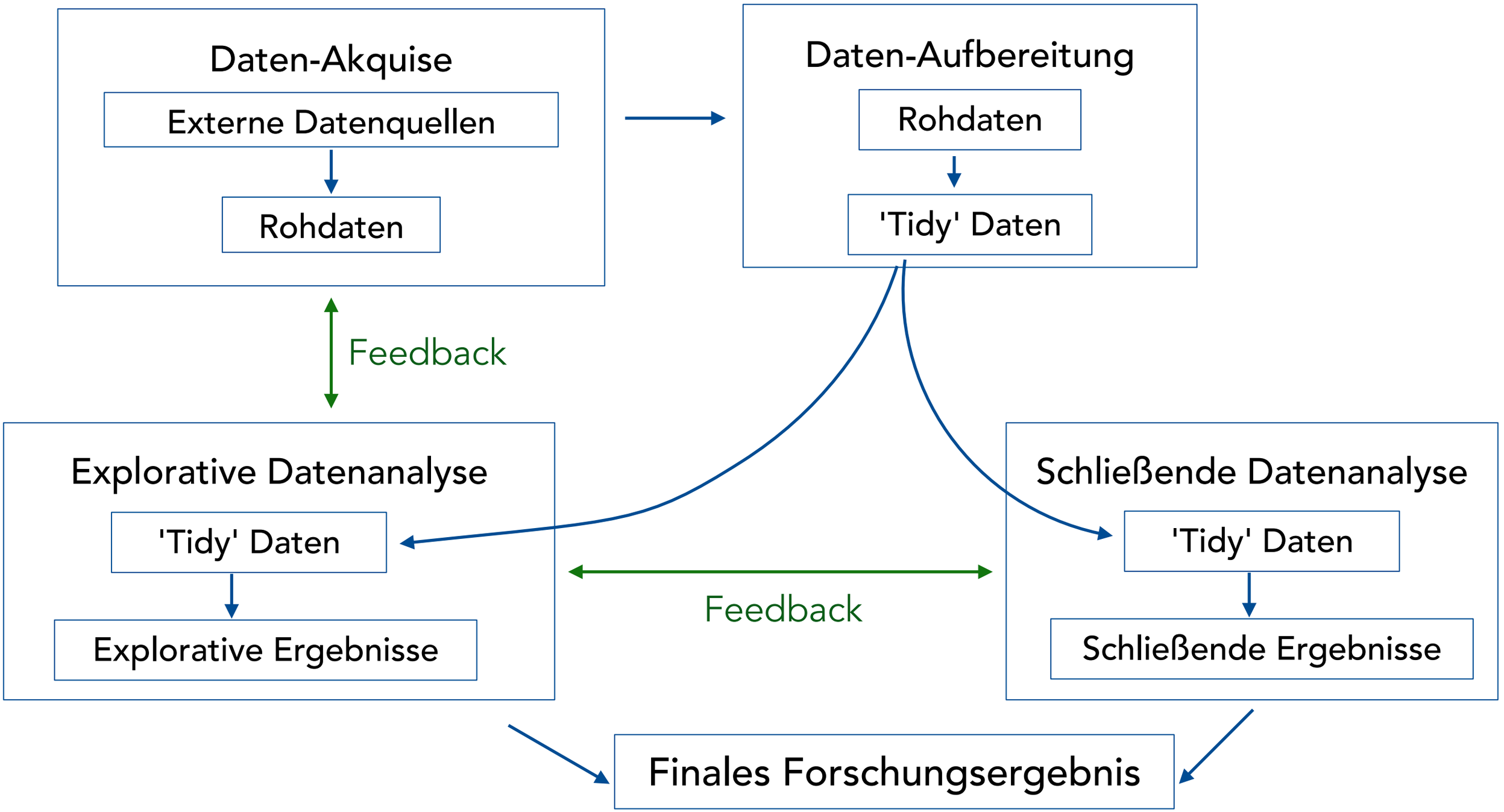
Open-Minded

ifso institute for socio-economics

Agenda

- Transparente, offene, reproduzierbare Forschung
- Datenkunde und Datenaufbereitung
  - Arten von Daten
  - Datenakquise
  - Einlesen & Schreiben von Datensätzen
  - Data wrangling
- Datenvisualisierung in R
  - Grundprinzip von ggplot2
  - Eine Beispielvisualisierung

Zum Ablauf eines empirischen Forschungsprojektes



Motivation: stuff that works…

- Laut einer aktuellen Umfrage verbringen Datenspezialisten ca. 80% ihrer Arbeitszeit mit der Erhebung und insbesondere der Aufbereitung von Daten.
- Diese beiden Schritte sind häufig der große Flaschenhals in der empirischen Forschung
  - Gut über die relevanten Tools Bescheid zu wissen…
  - …spart Zeit
  - …spart Nerven
  - …wird Ihnen viele neue Freund*innen bescheren

- Gerade die Datenaufbereitung hat ungeahnt breite Anwendungsfelder
  - Daumenregel: „Alles was man in Excel machen kann (und das Funktionen braucht), macht man vermutlich besser in R.

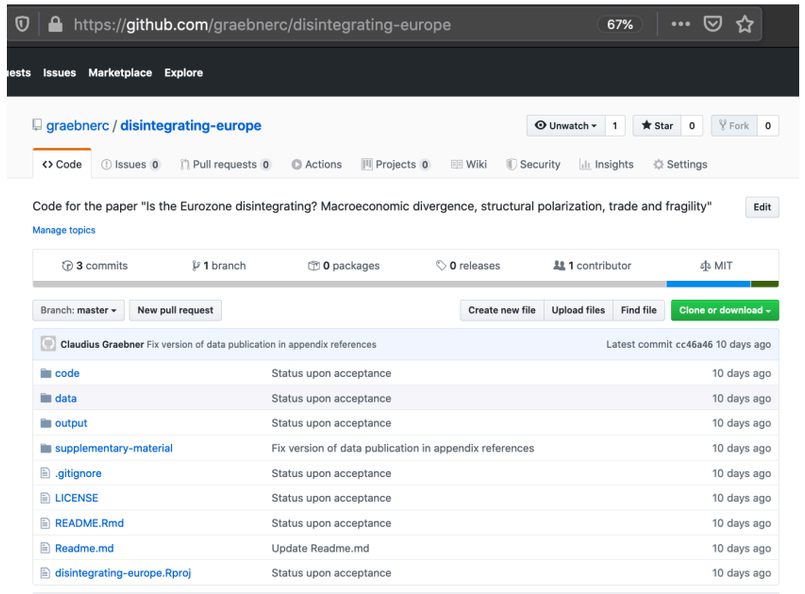
Motivation: Transparente Forschung

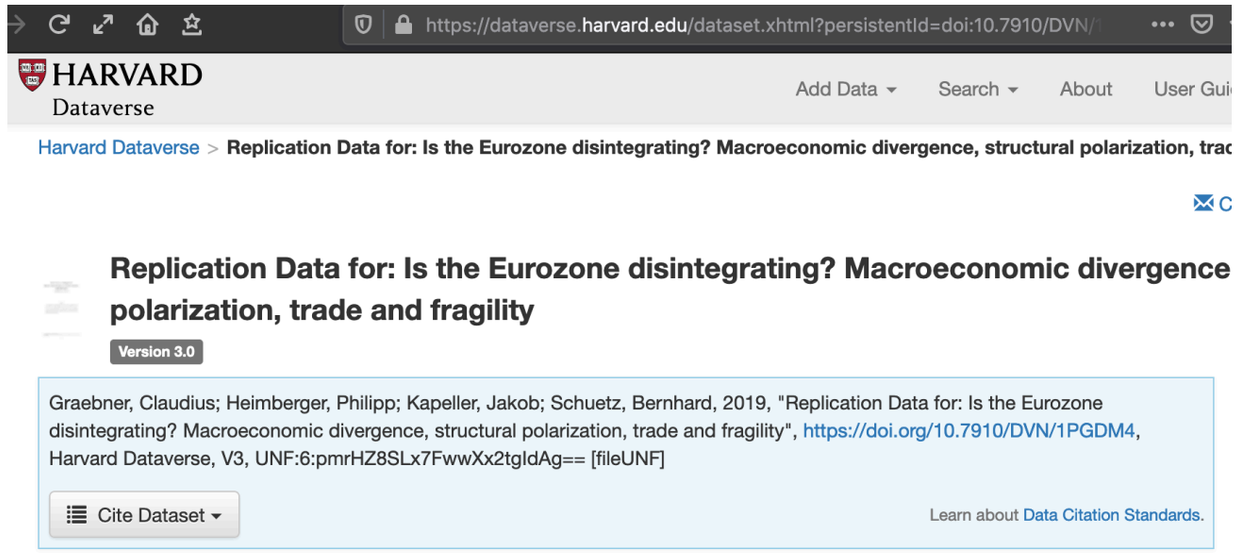
- Grundprinzip von 'open science': Jedes Forschungsergebnis muss von Grund auf reproduzierbar sein
  - "Von Grund auf" ~ "aus den Rohdaten"
- Implikationen:
  - Sie dürfen nie Ihre Rohdaten manipulieren
  - Sie müssen alle Arbeitsschritte nach Erhebung der Rohdaten dokumentieren

- Tatsächlich machen die Schritte auch die Kollaboration mit Ihren Kolleg*innen und Ihrem zukünftigen Ich deutlich einfacher

Hinweise zur transparenten Forschung

- Github: offener Code und Versionskontrolle
- Daten-Repositories: Veröffentlichung von Daten





Readme.md

Is the Eurozone disintegrating? Macroeconomic divergence, structural polarization, trade and fragility

Claudius Gräbner, Philipp Heimberger, Jakob Kapeller and Bernhard Schütz

Here we provide all the data and the code to replicate all empirical exercises in the abovementioned paper, which has been published in the *Cambridge Journal of Economics* (DOI: 10.1093/cje/bez059, Gräbner *et al.* 2019a).

The repository contains all code to replicate all figures in the paper. While all intermediate data required can be found in the repository, some raw data has not been deployed because the data is too large. However, from the source files the way we process the raw data becomes clear, and it is easy to use these files to recreate our reduced data if you are willing to download the raw data from the Atlas of Economic Complexity. There is also an accompanying data publication at the Harvard Dataverse, (Gräbner *et al.* 2019a), which contains all raw data.

Supported by funds of the Oesterreichische Nationalbank (Austrian Central Bank, Anniversary Fund, project number: 17383). The data as well as all code required to replicate the empirical exercises in the paper are available on Github: https://github.com/graebnerc/structural-change. The raw data are also published as (Gräbner et al. 2019).

Hinweise zur transparenten Forschung

Learning from past mistakes…

GROWTH IN A TIME OF DEBT

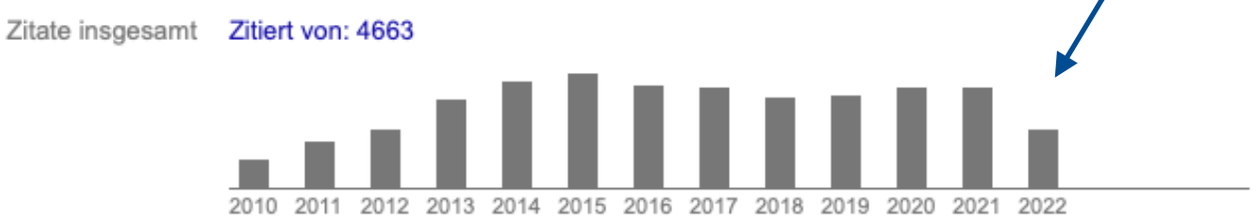
Carmen M. Reinhart
Kenneth S. Rogoff

Working Paper 15639
http://www.nber.org/papers/w15639

Punchline:
If Debt / GDP > 90%, this is very, very bad for growth

Learnings are possible!

Scaling of learnings is unsure ;-)



Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff

Get access

Thomas Herndon, Michael Ash, Robert Pollin

Cambridge Journal of Economics, Volume 38, Issue 2, March 2014, Pages 257–279,
https://doi.org/10.1093/cje/bet075
Published: 24 December 2013 **Article history**

Cite Permissions Share

Abstract

We replicate Reinhart and Rogoff (2010A and 2010B) and find that selective exclusion of available data, coding errors and inappropriate weighting of summary statistics lead to serious miscalculations that inaccurately represent the relationship between public debt and GDP growth among 20 advanced economies. Over 1946–2009, countries with public debt/GDP ratios above 90% averaged 2.2% real annual GDP growth, not −0.1% as published. The published results for (i) median GDP growth rates for the 1946–2009 period and (ii) mean and median GDP growth figures over 1790–2009 are all distorted by similar methodological errors, although the magnitudes of the distortions are somewhat smaller than with the mean figures for 1946–2009. Contrary to Reinhart and Rogoff's broader contentions, both mean and median GDP growth when public debt levels exceed 90% of GDP are not dramatically different from when the public debt/GDP ratios are lower. The relationship between public debt and GDP growth varies significantly by period and country. Our overall evidence refutes RR's claim that public debt/GDP ratios above 90% consistently reduce a country's GDP growth.

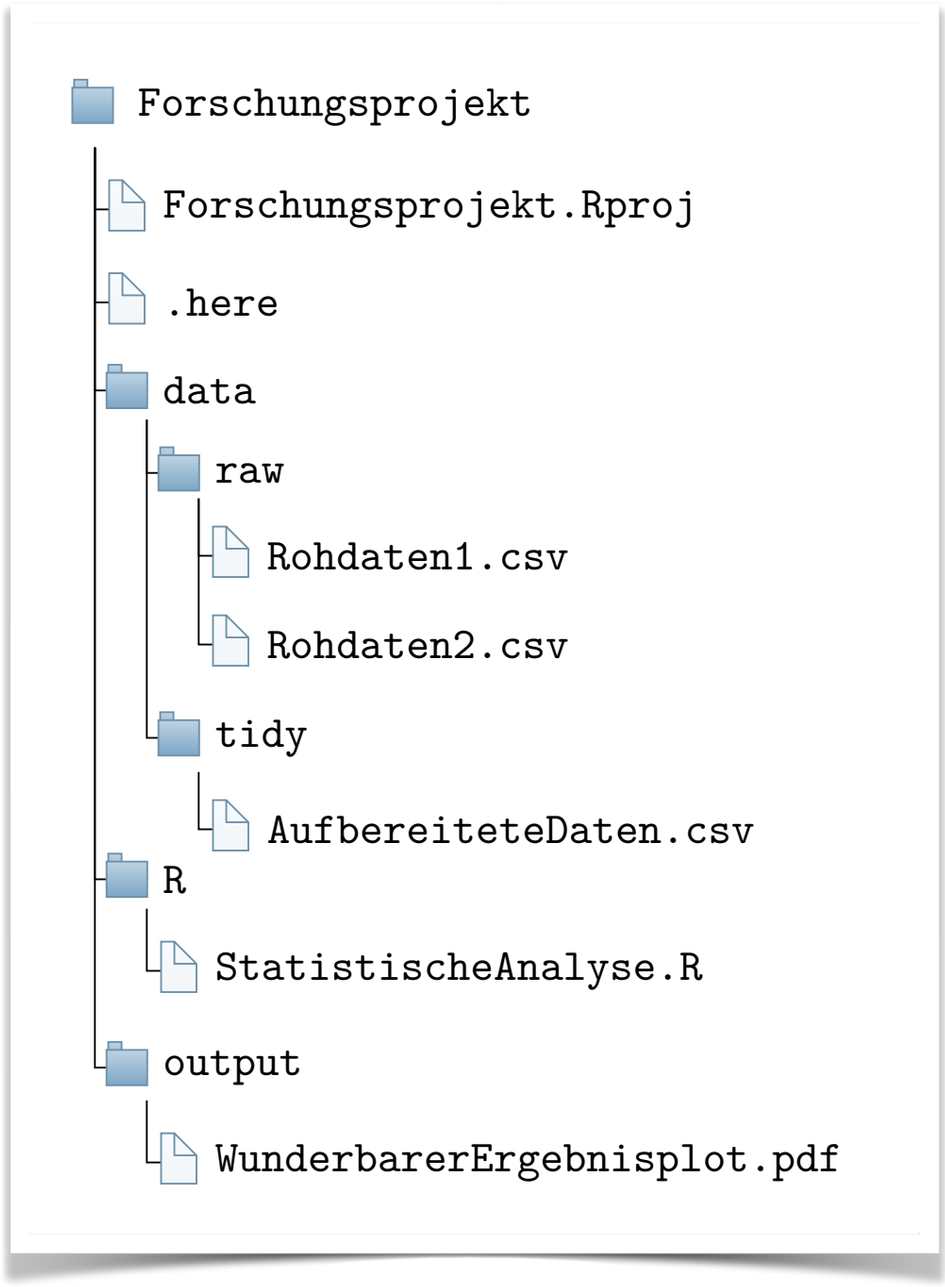
JEL: E60 - General, E62 - Fiscal Policy, E65 - Studies of Particular Policy Episodes

Issue Section: Article

Transparenz in der Praxis…

Organisation ihrer Forschungsarbeit

- Hier gehen wir davon aus, dass Sie Ihr Forschungsprojekt folgendermaßen organisieren:
- Vorteile:
  - Der output ist immer komplett aus den Rohdaten reproduzierbar
  - Der Code kann - wenn richtig geschrieben - beliebig zwischen Computern ausgetauscht werden
  - Es ist unmittelbar ersichtlich wie das Projekt aufgebaut ist
- Nachteile:
  - Verlangt eine gewisse Selbstdisziplin



Datenkunde und Datenaufbereitung

Arten von Daten

- Es gibt viele Klassifikationsschema für Daten
  - Quantitativ vs. Qualitativ (Welche Form?)
  - Manifest vs. Latent (direkt beobachtbar?)
  - Skalenniveau (Welche numerische Interpretation?)
- Letzteres praktisch besonders relevant: bestimmt Anwendbarkeit von statistischen Methoden:

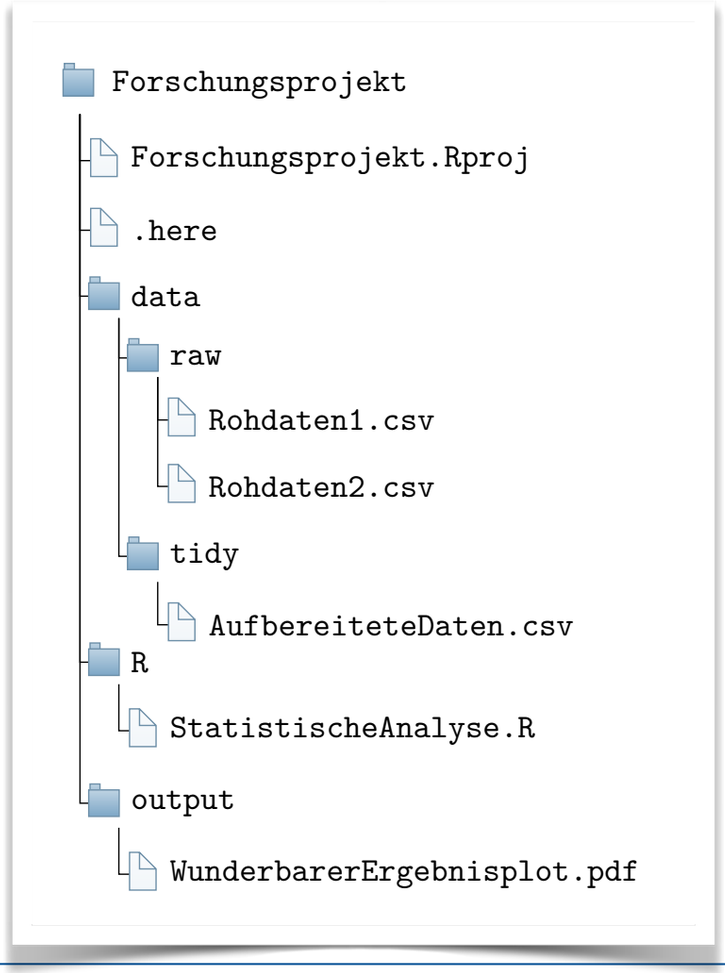
Arten von Daten

- Es gibt viele Klassifikationsschema für Daten
  - Quantitativ vs. Qualitativ (Welche Form?)
  - Manifest vs. Latent (direkt beobachtbar?)
  - Skalenniveau (Welche numerische Interpretation?)
- Letzteres praktisch besonders relevant: bestimmt Anwendbarkeit von statistischen Methoden:

Skalenniveau	Beispiel	Messbare Eigenschaften	Welches R Objekt?
Nominal	Haarfarbe, Telefonnummer	Häufigkeit	character, factor
Ordinal	Schulnote, Zufriedenheit	Häufigkeit, Rangfolge	factor
Intervall	Jahreszahl, Grad Celsius	Häufigkeit, Rangfolge, Abstand	integer, double
Verhältnis	Preise, Alter, Grad Kelvin	Häufigkeit, Rangfolge, Abstand, abs. Nullpunkt	integer, double

Datenakquise

- Primärerhebung: Sie produzieren Ihre Daten selbst
  - Machen Sie es sich einfach: schon bei der Akquise darauf achten welche Form die Daten für die spätere Analyse haben müssen
- Sekundärerhebung: Sie besorgen sich Daten von anderen
  - Typische Verdächtige: Weltbank, AMECO, OECD, UN, ...
- Meist sind die Daten in einem Zustand, der keine unmittelbare Analyse erlaubt
  - Ausnahme: Sie besorgen sich die Daten direkt über R (siehe Skript)
- Zentral: nach Erhebung die Metadaten speichern und die Daten nie mehr ändern



Daten einlesen

Formate, Formate, Formate…

- Disclaimer: Daten einlesen ist mühsamer als man denkt (→ Frust)…
  - … also planen Sie Zeit ein und versuchen Sie Ihre Erwartungshaltung zu moderieren ;-)
- Idee immer: Sie lesen die Daten ein und binden sie an ein Objekt, i.d.R. einen data.frame
- In R gibt es viele Funktionen, die Daten einlesen können
- Hängt v.a. vom Format der einzulesenden Daten ab

csv, tsv-Dateien

Gängige, universelle Formate

```
library(data.table)
fread()
```

.rds und .rdata

R-eigene Formate

```
readRDS()
load()
```

Andere Dateiformate

Andere (oft: proprietäre) Dateiformate

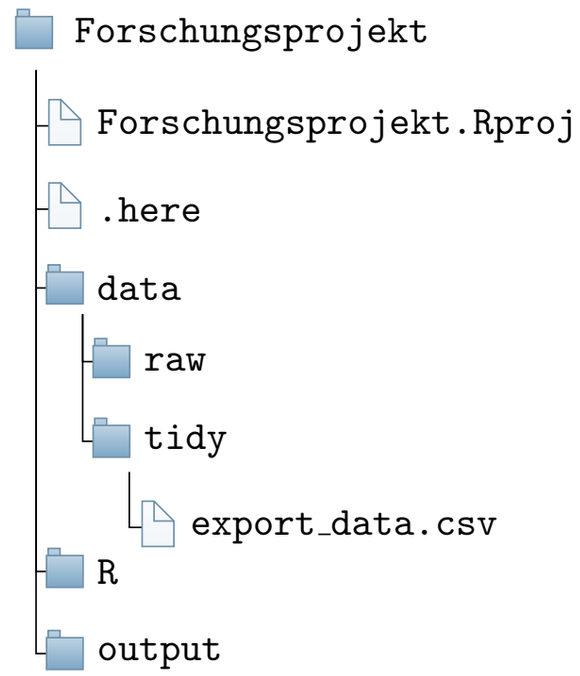
```
library(haven)
read_dta()
read_sas()
read_spss()
```

Daten einlesen

Beispiel: csv-Dateien einlesen

- Gute Praxis: zunächst den Dateipfad als Vektor speichern:

```
datei_pfad <- here("data/tidy/export_data.csv")
```



- Dann mit Hilfe von **fread()** einlesen:

```
datei_pfad <- here("data/tidy/export_data.csv")
export_daten <- fread(datei_pfad)
head(export_daten)
```

```
##                 cgroup commoditycode         pci    exp_share
## 1:      Core countries           101  0.06424262 0.0001312370
## 2: Periphery countries           101  0.06424262 0.0004639794
## 3:      Core countries           102 -0.49254290 0.0005162508
## 4: Periphery countries           102 -0.49254290 0.0003700469
## 5:      Core countries           103  0.51082386 0.0005324995
## 6: Periphery countries           103  0.51082386 0.0004082251
```

- Wichtigstes Argument für **fread()**: Pfad zur Datei
- Es ist aber sehr empfehlenswert einige zusätzliche Argumente zu verwenden

Daten einlesen

Beispiel: csv-Dateien einlesen

- Es ist meistens besser die Spalten-Typen selbst festzulegen, denn:

```
datei_pfad <- here("data/tidy/export_data.csv")
export_daten <- fread(datei_pfad)
head(export_daten)
```

```
##                 cgroup commoditycode         pci     exp_share
## 1:      Core countries           101  0.06424262 0.0001312370
## 2: Periphery countries           101  0.06424262 0.0004639794
## 3:      Core countries           102 -0.49254290 0.0005162508
## 4: Periphery countries           102 -0.49254290 0.0003700469
## 5:      Core countries           103  0.51082386 0.0005324995
## 6: Periphery countries           103  0.51082386 0.0004082251
```

```
datei_pfad <- here("data/tidy/export_data.csv")
export_daten <- fread(datei_pfad,
                      colClasses = c("character", "character",
                                     "double", "double")
                      )
head(export_daten)
```

```
##                 cgroup commoditycode         pci     exp_share
## 1:      Core countries          0101  0.06424262 0.0001312370
## 2: Periphery countries          0101  0.06424262 0.0004639794
## 3:      Core countries          0102 -0.49254290 0.0005162508
## 4: Periphery countries          0102 -0.49254290 0.0003700469
## 5:      Core countries          0103  0.51082386 0.0005324995
## 6: Periphery countries          0103  0.51082386 0.0004082251
```

```
typeof(export_daten[["commoditycode"]])
```

```
## [1] "integer"
```

- Die Produktcodes sind aber lediglich numerische Codes, die nicht als Zahlen sondern Wörter zu interpretieren sind

```
typeof(export_daten[["commoditycode"]])
```

```
## [1] "character"
```

- Bei der Interpretation als integer werden die vorangestellten Nullen gelöscht!

Daten einlesen

Beispiel: csv-Dateien einlesen

- Anderes Problem: ihre Daten verfügen über 'merkwürdige' Spalten- oder Dezimaltrennzeichen:

Angelsächsischer Standard:

```
iso2c,year,Exporte
AT,2012,53.97
```

```
daten_pfad <- here("data/tidy/export_daten.csv")
daten <- fread(daten_pfad)
daten
```

```
#>    iso2c year Exporte
#> 1:    AT 2012   53.97
```

Die 'deutsche Variante':

```
iso2c;year;Exporte
AT;2012;53,97
```

```
daten_pfad <- here("data/tidy/export_daten_dt.csv")
daten <- fread(daten_pfad,
               colClasses = c("character", "double", "double"),
               sep = ";",
               dec = ","
               )
daten
```

```
#>    iso2c year Exporte
#> 1:    AT 2012   53.97
```

- Weitere hilfreiche Argumente von **fread()**, z.B. zum selektiven Einlesen bestimmter Zeilen oder Spalten, werden im Skript eingeführt

Recently on my desktop…

- Voller Vorfreude werden Daten eingelesen.

```r
GDPWB_raw <- fread(here::here("raw data/Worldbank1.csv"), dec=",")
```

- Dann die fatale Erkenntnis, dass das Ergebnis so aussieht:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	Country Name	1960.0000	1961.0000	1962.0000	1963.0000	1964.0000	1965.0000	1966.0000	1967.0000	1968.0000
2	Aruba	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
3	Africa Eastern and Southern	1175.7353	1149.3606	1209.9305	1240.0882	1263.6101	1296.4906	1312.0850	1344.7552	1361.7142
4	Afghanistan	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
5	Africa Western and Central	1087.6399	1085.1126	1102.4994	1157.5209	1193.6896	1214.9844	1167.0360	1032.1586	1023.7001
6	Angola	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
7	Albania	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
8	Andorra	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
9	Arab World	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
10	United Arab Emirates	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
11	Argentina	7362.5341	7637.0667	7451.8034	6945.9571	7532.0045	8202.1125	8026.8762	8161.6021	8429.8689
12	Armenia	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
13	American Samoa	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*

- Kein Bock auf Checken der Einleseoptionen?
- Dann einfach durchkneten, bis es tidy ist!

````
```{r data_wrangling, echo=TRUE}
GDPWB <- tidyr::pivot_longer(
  data = GDPWB_raw,
  cols = -any_of(c("V1")),
  names_to = "Jahr",
  values_to = "GDP") %>%
    dplyr::rename(., Country=V1) %>%
    dplyr::mutate(Jahr=as.double(stringr::str_remove(Jahr,"V"))+1958) %>%
    dplyr::filter(Country!="Country Name")
```
````

Speichern von Daten

- Speichern ist einfacher als einlesen.
- Relevant eigentlich nur die Frage nach dem Format
  - Universal choice: csv als offenes, simples und weitverbreitetes Format.
  - Probably most comfortable choice: Die R-internen Formate haben den Vorteil, dass diese sich merken um welche Datentypen es sich handelt. Lässige Usability in R.
  - Für höher dimensionierte Daten: JSON als offenes, simples und weitverbreitetes Format – man kann es als großen Bruder von csv sehen.

```
datei_name <- here("data/tidy/test_data.csv")
fwrite(test_data, file = datei_name)
```

- Für Details siehe das Skript

Data wrangling

- Beim 'data wrangling' geht es darum die Rohdaten in eine Form zu bringen, mit der man dann gut weiterarbeiten kann
  - Visualisierungen gestalten
  - Statistische Verfahren anwenden
- Gerade hier ist Dokumentation über Skripte das A und O!
- Hier betrachten wir einige typische Herausforderungen:
  - Lange und breite Daten
  - Mehrere Datensätze kombinieren
  - Daten filtern, selektieren und zusammenfassen
- Dabei verwenden wir eine Sammlung von Paketen, das **tidyverse**
  - Siehe Kommentare im Skript
- Doch zuvor: wie sollen unsere Daten am Ende eigentlich aussehen?

Das Ziel von data wrangling: 'tidy data'

‚tidy data' als Ausgangspunkt für alle weiteren Schritte

- Am Ende des Aufbereitungsprozesses sollen die Daten 'tidy' sein:

1. Jede Spalte korrespondiert zu genau einer Variable

2. Jede Zeile korrespondiert zu genau einer Beobachtung

```
#>   Land Jahr  Exporte Arbeitslosigkeit
#> 1   AT 2013 53.44129            5.335
#> 2   AT 2014 53.38658            5.620
#> 3   DE 2013 45.39788            5.231
#> 4   DE 2014 45.64482            4.981
```

3. Jede Zelle korrespondiert zu genau einem Wert

- Beispiele für ungeordnete Daten:

```
  Land          Variable     2013     2014
1   AT Arbeitslosigkeit  5.33500  5.62000
2   AT          Exporte 53.44129 53.38658
3   DE Arbeitslosigkeit  5.23100  4.98100
4   DE          Exporte 45.39788 45.64482
```

```
#>   Land Jahr          Variable     Wert
#> 1   AT 2013          Exporte 53.44129
#> 2   AT 2014          Exporte 53.38658
#> 3   DE 2013          Exporte 45.39788
#> 4   DE 2014          Exporte 45.64482
```

```
  Land   Wichtige Industrien
1   DE    Autos, Medikamente
2   AT Stahlproduktion, Holz
```

Daten kommen in vielen Formen und Formaten

Eine wichtige Unterscheidung: „breite“ und „lange“ Datensätze

- Daten, die Sie herunterladen sind häufig im - für Menschen besser lesbaren - "breiten" Format:

```
data_wide
```

```
#>    Land  2013  2014
#> 1    AT 5.335 5.620
#> 2    DE 5.231 4.981
```

breit

Von Spalten zu Zeilen: pivot_longer

Von Zeilen zu Spalten: pivot_wider

```
# A tibble: 4 x 3
  Land  Jahr  Arbeitslosenquote
  <chr> <chr>             <dbl>
1 AT    2013               5.34
2 AT    2014               5.62
3 DE    2013               5.23
4 DE    2014               4.98
```

lang

Zu transformierender Datensatz

```
data_long <- pivot_longer(data = data_wide,
                          cols = any_of(c("2013", "2014")),
                          names_to = "Jahr",
                          values_to = "Arbeitslosenquote")
```

Zu transformierende Spalte(n)

Name der neuen Spalte (Transformation)

Name der neuen Spalte (Übertragene Werte)

Daten kommen in vielen Formen und Formaten

Eine wichtige Unterscheidung: „breite“ und „lange“ Datensätze

- Den umgekehrten Weg gehen wir mit pivot_wider():
  - Achtung: Das ist eigentlich nicht ‚tidy‘, aber gut.

```
# A tibble: 4 x 3
  Land  Jahr  Arbeitslosenquote
  <chr> <chr>              <dbl>
1 AT    2013                5.34
2 AT    2014                5.62
3 DE    2013                5.23
4 DE    2014                4.98
```

Von Spalten zu Zeilen: pivot_longer

```
# A tibble: 2 x 3
  Land  `2013` `2014`
  <chr>  <dbl>  <dbl>
1 AT      5.34   5.62
2 DE      5.23   4.98
```

Zu transformierender Datensatz

```
data_wide_neu <- pivot_wider(data = data_long,
                             id_cols = any_of("Land"),
                             names_from = "Jahr",
                             values_from = "Arbeitslosenquote")
```

Name der ersten Spalte

Name der weiteren Spalten

Name der Spalte mit den zu übertragenden Werten

Nicht alles was zusammengehört, ist schon zusammen ;-)

Oft muss man Daten aus verschiedenen Quellen zusammenführen!

- Situation: Sie haben Daten aus zwei Quellen erhoben, z.B. Gini Daten aus der SWIID und BIP Daten von der Weltbank, und wollen diese kombinieren

```
  Jahr Land BIP
1 2010  DEU   1
2 2011  DEU   2
3 2012  DEU   3
4 2010  AUT   4
5 2011  AUT   5
6 2012  AUT   6
```

+

```
  year country Gini
1 2010     DEU    1
2 2011     DEU    2
3 2012     AUT    3
4 2013     AUT    4
```

→

```
  Jahr Land BIP Gini
1 2010  DEU   1    1
2 2011  DEU   2    2
3 2012  DEU   3   NA
4 2010  AUT   4   NA
5 2011  AUT   5   NA
6 2012  AUT   6    3
```

o.ä.

- Je nachdem was wir als Ergebnis haben wollen verwenden wir eine der *_join()-Funktionen:

Nicht alles was zusammengehört, ist schon zusammen ;-)

Oft muss man Daten aus verschiedenen Quellen zusammenführen!

- Situation: Sie haben Daten aus zwei Quellen erhoben, z.B. Gini Daten aus der SWIID und BIP Daten von der Weltbank, und wollen diese kombinieren

	Jahr	Land	BIP
1	2010	DEU	1
2	2011	DEU	2
3	2012	DEU	3
4	2010	AUT	4
5	2011	AUT	5
6	2012	AUT	6

+

	year	country	Gini
1	2010	DEU	1
2	2011	DEU	2
3	2012	AUT	3
4	2013	AUT	4

→

	Jahr	Land	BIP	Gini
1	2010	DEU	1	1
2	2011	DEU	2	2
3	2012	DEU	3	NA
4	2010	AUT	4	NA
5	2011	AUT	5	NA
6	2012	AUT	6	3

o.ä.

- Je nachdem was wir als Ergebnis haben wollen verwenden wir eine der *_join()-Funktionen:
  1. left_join(): b in a integrieren (neue NA nur in b-Spalten)

Nicht alles was zusammengehört, ist schon zusammen ;-)

Oft muss man Daten aus verschiedenen Quellen zusammenführen!

- Situation: Sie haben Daten aus zwei Quellen erhoben, z.B. Gini Daten aus der SWIID und BIP Daten von der Weltbank, und wollen diese kombinieren

```
  Jahr Land BIP
1 2010  DEU   1
2 2011  DEU   2
3 2012  DEU   3
4 2010  AUT   4
5 2011  AUT   5
6 2012  AUT   6
```

+

```
  year country Gini
1 2010     DEU    1
2 2011     DEU    2
3 2012     AUT    3
4 2013     AUT    4
```

→

```
##   Jahr Land BIP Gini
## 1 2010  DEU   1    1
## 2 2011  DEU   2    2
## 3 2012  AUT   6    3
## 4 2013  AUT  NA    4
```

o.ä.

- Je nachdem was wir als Ergebnis haben wollen verwenden wir eine der *_join()-Funktionen:
  1. left_join(): b in a integrieren (neue NA nur in b-Spalten)
  2. right_join(): a in b integrieren (neue NA nur in a-Spalten)

Nicht alles was zusammengehört, ist schon zusammen ;-)

Oft muss man Daten aus verschiedenen Quellen zusammenführen!

- Situation: Sie haben Daten aus zwei Quellen erhoben, z.B. Gini Daten aus der SWIID und BIP Daten von der Weltbank, und wollen diese kombinieren

```
  Jahr Land BIP
1 2010  DEU   1
2 2011  DEU   2
3 2012  DEU   3
4 2010  AUT   4
5 2011  AUT   5
6 2012  AUT   6
```

+

```
  year country Gini
1 2010     DEU    1
2 2011     DEU    2
3 2012     AUT    3
4 2013     AUT    4
```

→

```
#> 1 2010  DEU   1    1
#> 2 2011  DEU   2    2
#> 3 2012  AUT   6    3
```

o.ä.

- Je nachdem was wir als Ergebnis haben wollen verwenden wir eine der *_join()-Funktionen:
  1. left_join(): b in a integrieren (neue NA nur in b-Spalten)
  2. right_join(): a in b integrieren (neue NA nur in a-Spalten)
  3. inner_join(): in beiden vorhandene Beobachtungen an a anhängen (keine neuen NA)

Nicht alles was zusammengehört, ist schon zusammen ;-)

Oft muss man Daten aus verschiedenen Quellen zusammenführen!

- Situation: Sie haben Daten aus zwei Quellen erhoben, z.B. Gini Daten aus der SWIID und BIP Daten von der Weltbank, und wollen diese kombinieren

```
  Jahr Land BIP
1 2010  DEU   1
2 2011  DEU   2
3 2012  DEU   3
4 2010  AUT   4
5 2011  AUT   5
6 2012  AUT   6
```

+

```
  year country Gini
1 2010     DEU    1
2 2011     DEU    2
3 2012     AUT    3
4 2013     AUT    4
```



```
#>   Jahr Land BIP Gini
#> 1 2010  DEU   1    1
#> 2 2011  DEU   2    2
#> 3 2012  DEU   3   NA
#> 4 2010  AUT   4   NA
#> 5 2011  AUT   5   NA
#> 6 2012  AUT   6    3
#> 7 2013  AUT  NA    4
```

o.ä.

- Je nachdem was wir als Ergebnis haben wollen verwenden wir eine der *_join()-Funktionen:
  1. left_join(): b in a integrieren (neue NA nur in b-Spalten)
  2. right_join(): a in b integrieren (neue NA nur in a-Spalten)
  3. inner_join(): in beiden vorhandene Beobachtungen an a anhängen (keine neuen NA)
  4. full_join(): a und b kombinieren (neue NA auf beiden Seiten)

Next step: Filtern und Selektieren

Nach breit&lang und matching der dritte Schritt…

- Aufgabe: bestimmte Spalten aus dem Datensatz entfernen:

```
head(
  select(data_al_exp_tidy, Land, Exporte),
  2)
```

```
#> # A tibble: 2 x 2
#>   Land  Exporte
#>   <chr>   <dbl>
#> 1 AT       54.0
#> 2 AT       53.4
```

```
head(
  select(data_al_exp_tidy, -Exporte),
  2)
```

```
#> # A tibble: 2 x 3
#>   Land  Jahr  Arbeitslosigkeit
#>   <chr> <chr>            <dbl>
#> 1 AT    2012              4.86
#> 2 AT    2013              5.34
```

```
#> # A tibble: 6 x 4
#>   Land  Jahr  Exporte Arbeitslosigkeit
#>   <chr> <chr>   <dbl>            <dbl>
#> 1 AT    2012     54.0             4.86
#> 2 AT    2013     53.4             5.34
#> 3 AT    2014     53.4             5.62
#> 4 DE    2012     46.0             5.38
#> 5 DE    2013     45.4             5.23
#> 6 DE    2014     45.6             4.98
```

- Zwei Konventionen:
  - Mit Spaltennamen als character wird es übersichtlicher
  - Häufig besser %>% zu verwenden

```
data_al_exp_selected <- data_al_exp_tidy %>%
  select(any_of(c("Land", "Jahr", "Exporte")))
head(data_al_exp_selected, 2)
```

```
## # A tibble: 2 x 3
##   Land  Jahr  Exporte
##   <chr> <chr>   <dbl>
## 1 AT    2012     54.0
## 2 AT    2013     53.4
```

Next step: Filtern und Selektieren

Nach breit&lang und matching der dritte Schritt…

- Aufgabe: bestimmte *Zeilen* aus dem Datensatz entfernen:

```
data_al_exp_filtered <- data_al_exp_tidy %>%
  filter(Land == "AT",
         Jahr > 2012)
data_al_exp_filtered
```

```
#> # A tibble: 2 x 4
#>   Land  Jahr  Exporte Arbeitslosigkeit
#>   <chr> <chr>   <dbl>             <dbl>
#> 1 AT    2013     53.4              5.34
#> 2 AT    2014     53.4              5.62
```

- Aufgabe: bestimmte *Spalten* aus dem Datensatz umbenennen:

```
#> # A tibble: 6 x 4
#>   Land  Jahr  Exporte Arbeitslosigkeit
#>   <chr> <chr>   <dbl>             <dbl>
#> 1 AT    2012     54.0              4.86
#> 2 AT    2013     53.4              5.34
#> 3 AT    2014     53.4              5.62
#> 4 DE    2012     46.0              5.38
#> 5 DE    2013     45.4              5.23
#> 6 DE    2014     45.6              4.98
```

```
data_al_exp_tidy %>%
  rename(country=Land,
         year_observation=Jahr,
         exports=Exporte,
         unemployment=Arbeitslosigkeit)
```

```
#> # A tibble: 6 x 4
#>   country year_observation exports unemployment
#>   <chr>   <chr>                <dbl>        <dbl>
#> 1 AT      2012                  54.0         4.86
```

Von den bearbeiteten Rohdaten zur Anwendung…

Sehr oft muss man sich passende Reihe erst generieren

```
head(unemp_data_wb)
```

```
#>    country year laborforce_female workforce_total population_total
#> 1:      AT 2010          46.13933         4276558          8363404
#> 2:      AT 2011          46.33455         4305310          8391643
#> 3:      AT 2012          46.50653         4352701          8429991
#> 4:      AT 2013          46.57752         4394285          8479823
#> 5:      AT 2014          46.70688         4412800          8546356
#> 6:      AT 2015          46.67447         4460833          8642699
```

- Wenn Sie eine Spalte ändern wollen verwenden wir **mutate()**:

```
unemp_data_wb <- unemp_data_wb %>%
  mutate(
    country = countrycode(country, "iso2c", "iso3c")
    )
head(unemp_data_wb, 2)
```

```
#>   country year laborforce_female workforce_total population_total
#> 1     AUT 2010          46.13933         4276558          8363404
#> 2     AUT 2011          46.33455         4305310          8391643
```

- Mit dem gleichen Befehl können Sie eine Spalte hinzufügen

Von den bearbeiteten Rohdaten zur Anwendung...

Sehr oft muss man sich passende Reihe erst generieren

```
head(unemp_data_wb)

#>    country year laborforce_female workforce_total population_total
#> 1:      AT 2010          46.13933         4276558           8363404
#> 2:      AT 2011          46.33455         4305310           8391643
#> 3:      AT 2012          46.50653         4352701           8429991
#> 4:      AT 2013          46.57752         4394285           8479823
#> 5:      AT 2014          46.70688         4412800           8546356
#> 6:      AT 2015          46.67447         4460833           8642699
```

- Wenn Sie eine Spalte ändern wollen verwenden wir **mutate()**
- Mit dem gleichen Befehl können Sie eine Spalte hinzufügen

```
unemp_data_wb <- unemp_data_wb %>%
  mutate(
    workers_female_total = laborforce_female*workforce_total/100
    )
head(unemp_data_wb, 2)

#>   country year laborforce_female workforce_total population_total
#> 1     AUT 2010          46.13933         4276558           8363404
#> 2     AUT 2011          46.33455         4305310           8391643
#>   workers_female_total
#> 1              1973175
#> 2              1994846
```

Von den bearbeiteten Rohdaten zur Anwendung…

Sehr oft muss man sich passende Informationen erst extrahieren

```
  country year laborforce_female workforce_total population_total
1     AUT 2010          46.13933         4276558           8363404
2     AUT 2011          46.33455         4305310           8391643
  workers_female_total
1              1973175
2              1994846
```

- Wenn Sie den Datensatz zusammenfassen wollen verwenden sie summarize():

```
unemp_data_wb_summarized <- unemp_data_wb %>%
  summarise(
    fem_workers_avg = mean(workers_female_total)
    )
unemp_data_wb_summarized
```

```
#>   fem_workers_avg
#> 1        10761223
```

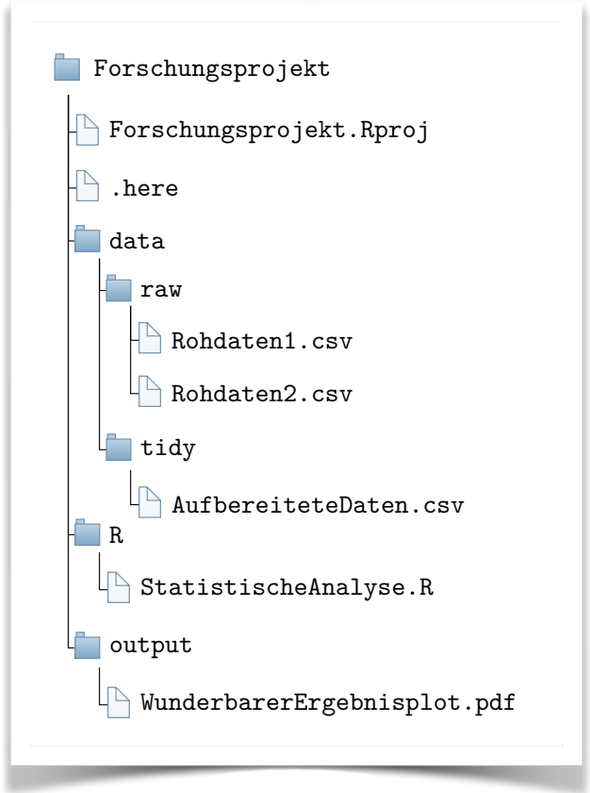
- Wenn Sie dabei den Datensatz gruppieren wollen, verwenden Sie group_by():

```
unemp_data_wb %>%
  group_by(country) %>%
  summarise(
    fem_workers_avg = mean(workers_female_total)
ungroup()
```

```
# A tibble: 2 x 2
  country fem_workers_avg
  <chr>             <dbl>
1 AUT             2042685.
2 DEU            19479761.
```

Zusammenfassung Datenaufbereitung:

- Der gesamte Weg von den Rohdaten hin zu den aufbereiteten Daten sollte immer transparent und nachvollziehbar sein
- Die erhobenen Daten sollten nie selbst manipuliert werden
- Daten sollten mit einem Skript aufgearbeitet und unter einem neuen Namen gespeichert werden
- Die Aufbereitung beinhaltet u.a.:
  - Einlesen der Daten
  - Schreiben von Daten
  - Daten selektieren
  - Daten zusammenfassen

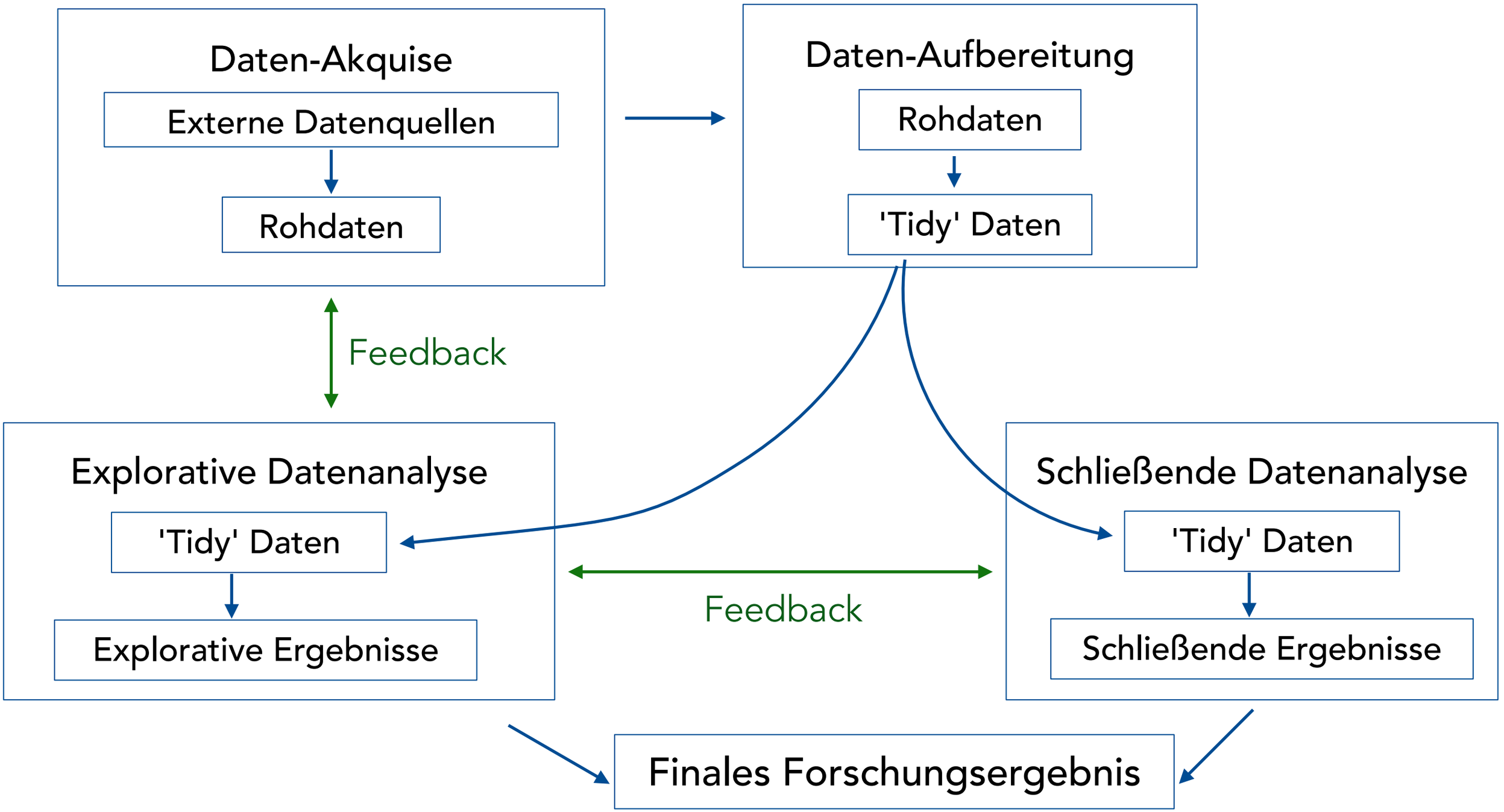


Datenvisualisierung in R

Teil 2: Datenvisualisierung

- Wir gehen jetzt davon aus, dass wir über geordnete ('tidy') Daten verfügen
  - Wir wollen Daten mit dem Paket ggplot2 aus dem tidyverse visualisieren
  - In ggplot2 werden Grafiken scheibchenweise gebaut, d.h. verschiedene *layer* werden übereinender gelegt
- Prinzip:
  - Sie erstellen eine komplexe Liste, die ihre Grafik beschreibt
  - Sie fügen zu der Liste so lange neue Details hinzu bis Sie die Grafik fertig beschrieben haben
  - Sie rufen die Liste auf, damit wird die finale Grafik erstellt
- Theoretischer Hintergrund: Die *Grammatik für Grafiken* (siehe Skript)
- Wir werden uns mit dieser Funktionsweise anhand eines Beispiels vertraut machen.

Zum Ablauf eines empirischen Forschungsprojektes

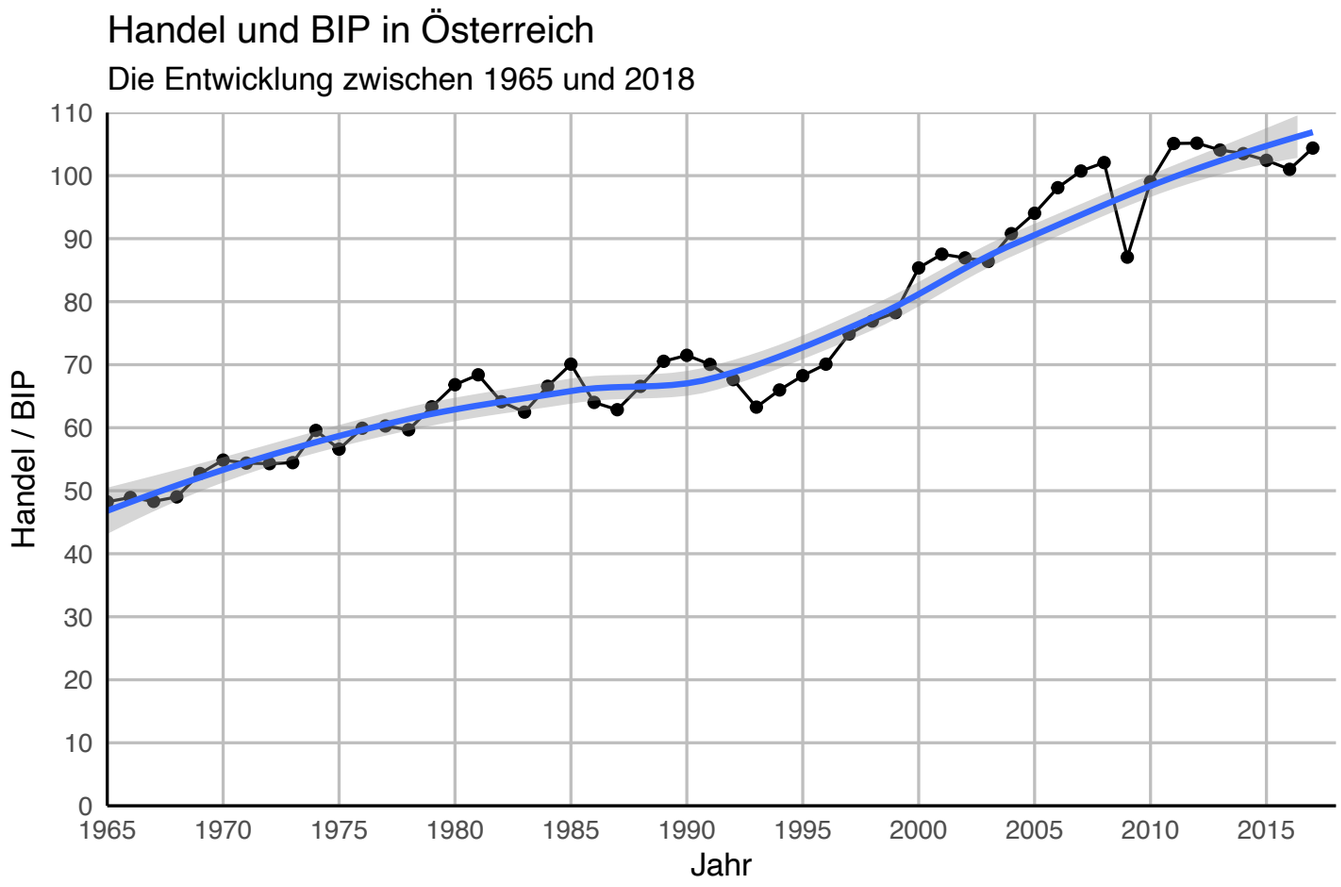


Datenvisualisierung mit ggplot2: ein Beispiel

- Ausgangsdaten:

```
#>    Land Jahr HandelGDP
#> 1   AUT 1965  48.23931
#> 2   AUT 1966  48.92554
#> 3   AUT 1967  48.30854
#> 4   AUT 1968  49.01388
#> 5   AUT 1969  52.72526
#> 6   AUT 1970  54.86039
```

- Gewünschtes Ergebnis:



Quelle: Weltbank.

- Wir stellen diese Abbildung Schritt für Schritt her

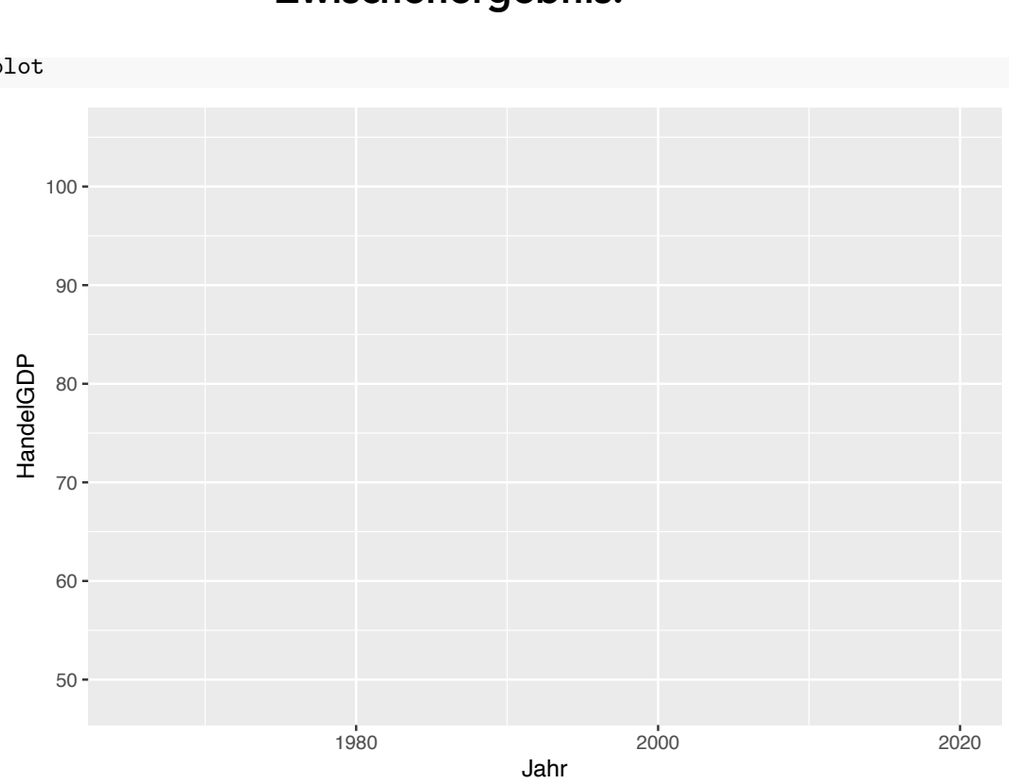
Datenvisualisierung mit ggplot2: ein Beispiel

- Zunächst erstellen wir eine besondere Liste, mit der die Grafik beschrieben wird:

```
aut_trade_plot <- ggplot(
  data = aut_trade,
  mapping = aes(x = Jahr,
                y = HandelGDP)
  )
```

Zwischenergebnis:

```
aut_trade_plot
```



Datenvisualisierung mit ggplot2: ein Beispiel

- Als nächstes fügen wir eine Linie als so genanntes *geom* hinzu:

```
aut_trade_plot <- ggplot(
  data = aut_trade,
  mapping = aes(x = Jahr,
                y = HandelGDP)
  )
```

```
aut_trade_plot <- aut_trade_plot +
  geom_line()
aut_trade_plot
```

Zwischenergebnis:

Datenvisualisierung mit ggplot2: ein Beispiel

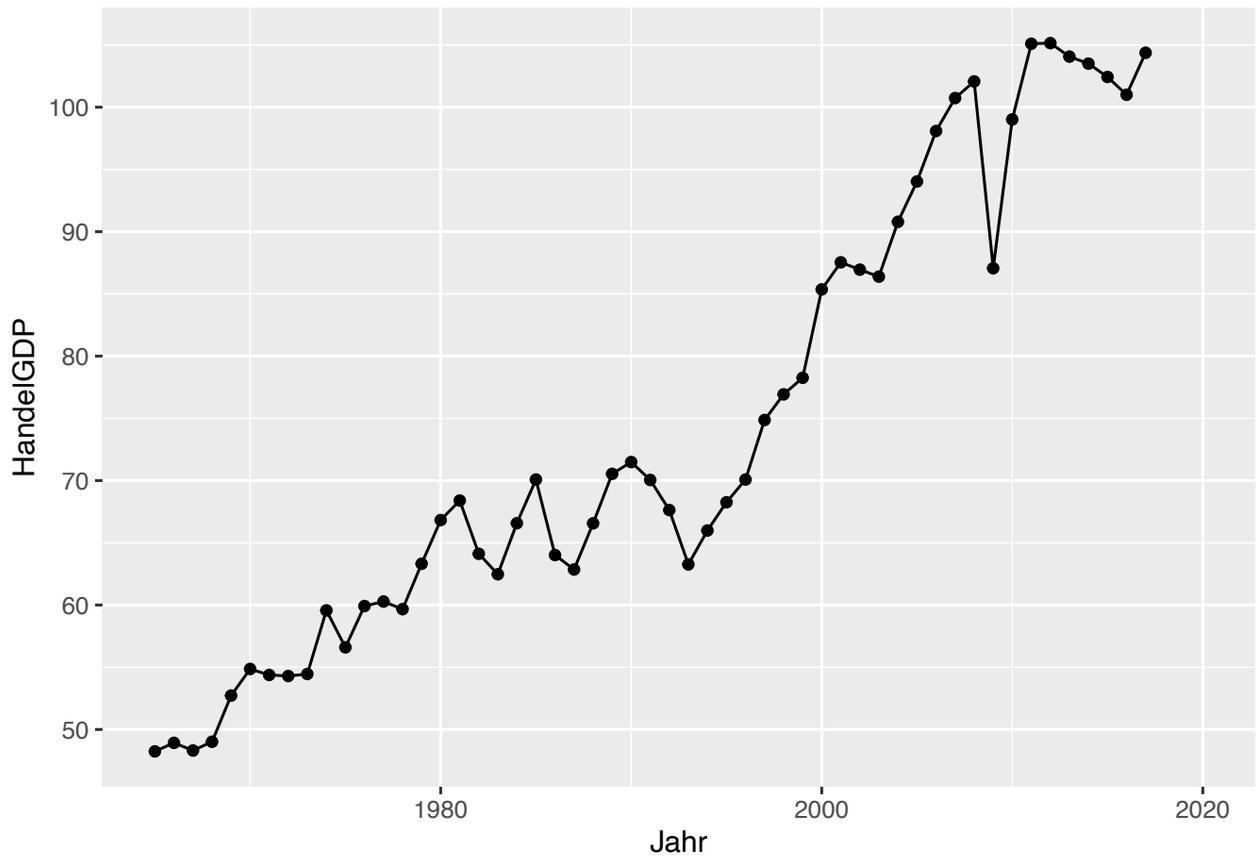
- Um die einzelnen Beobachtungen besser sehen zu können legen wir noch eine Ebene mit Punkten darüber:

```
aut_trade_plot <- ggplot(
  data = aut_trade,
  mapping = aes(x = Jahr,
                y = HandelGDP)
  )
```

```
aut_trade_plot <- aut_trade_plot +
  geom_line()
aut_trade_plot
```

```
aut_trade_plot <- aut_trade_plot +
  geom_point()
```

Zwischenergebnis:



Datenvisualisierung mit ggplot2: ein Beispiel

- Und schließlich noch eine weitere Ebene mit dem Trend:

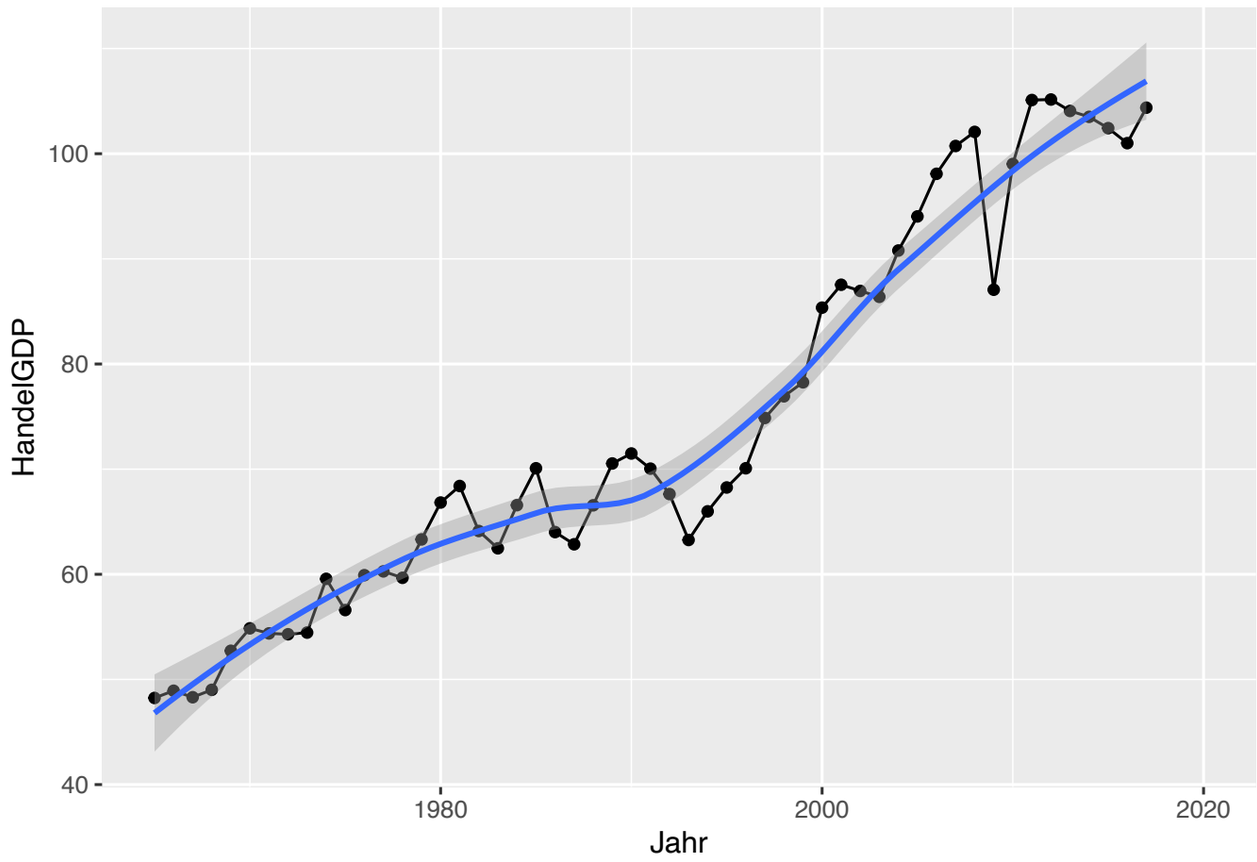
```
aut_trade_plot <- ggplot(
  data = aut_trade,
  mapping = aes(x = Jahr,
                y = HandelGDP)
  )
```

```
aut_trade_plot <- aut_trade_plot +
  geom_line()
```

```
aut_trade_plot <- aut_trade_plot +
  geom_point()
```

```
aut_trade_plot <- aut_trade_plot +
  geom_smooth()
```

Zwischenergebnis:

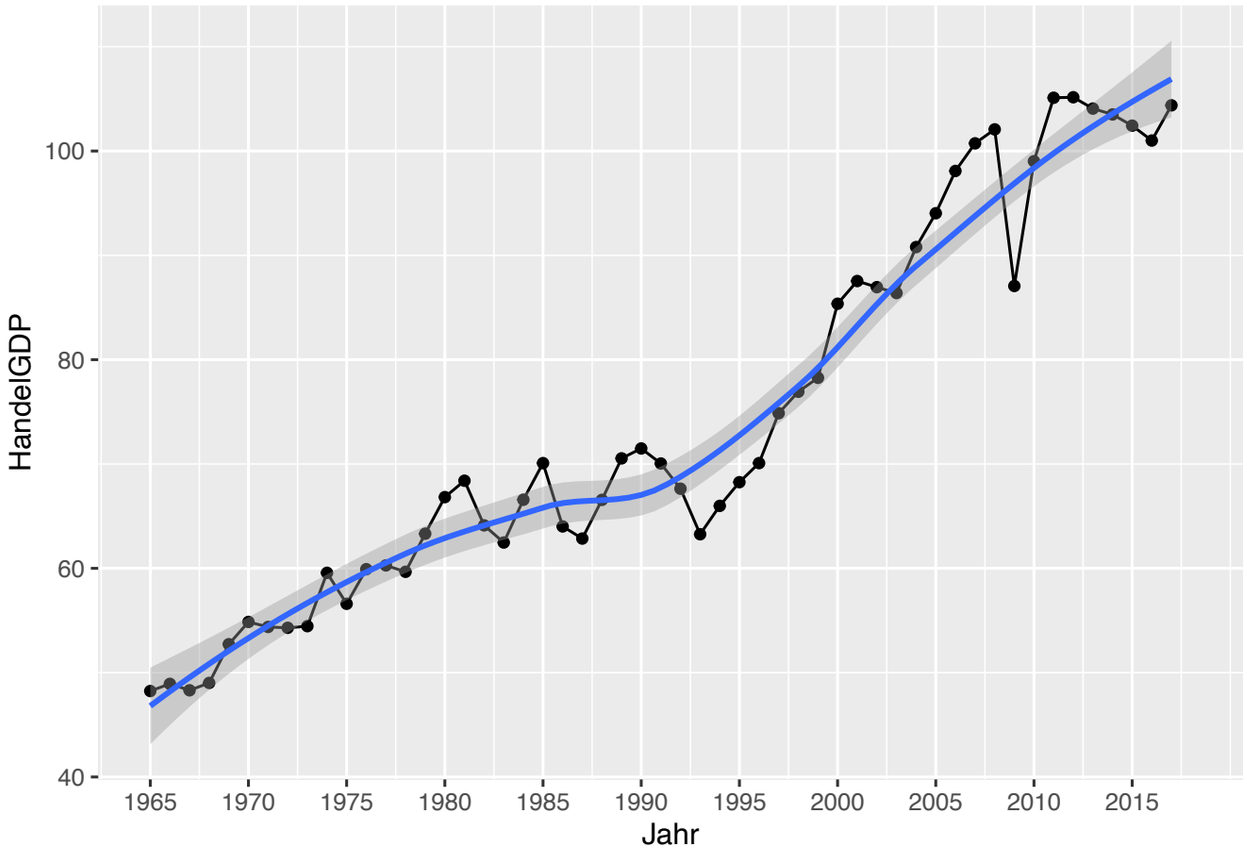


Datenvisualisierung mit ggplot2: ein Beispiel

- Nun wollen wir noch die Skalen anpassen:

```
aut_trade_plot <- aut_trade_plot +
  scale_x_continuous(limits = c(1965, 2018),
                     breaks = seq(1965, 2017, 5))
```

Zwischenergebnis:

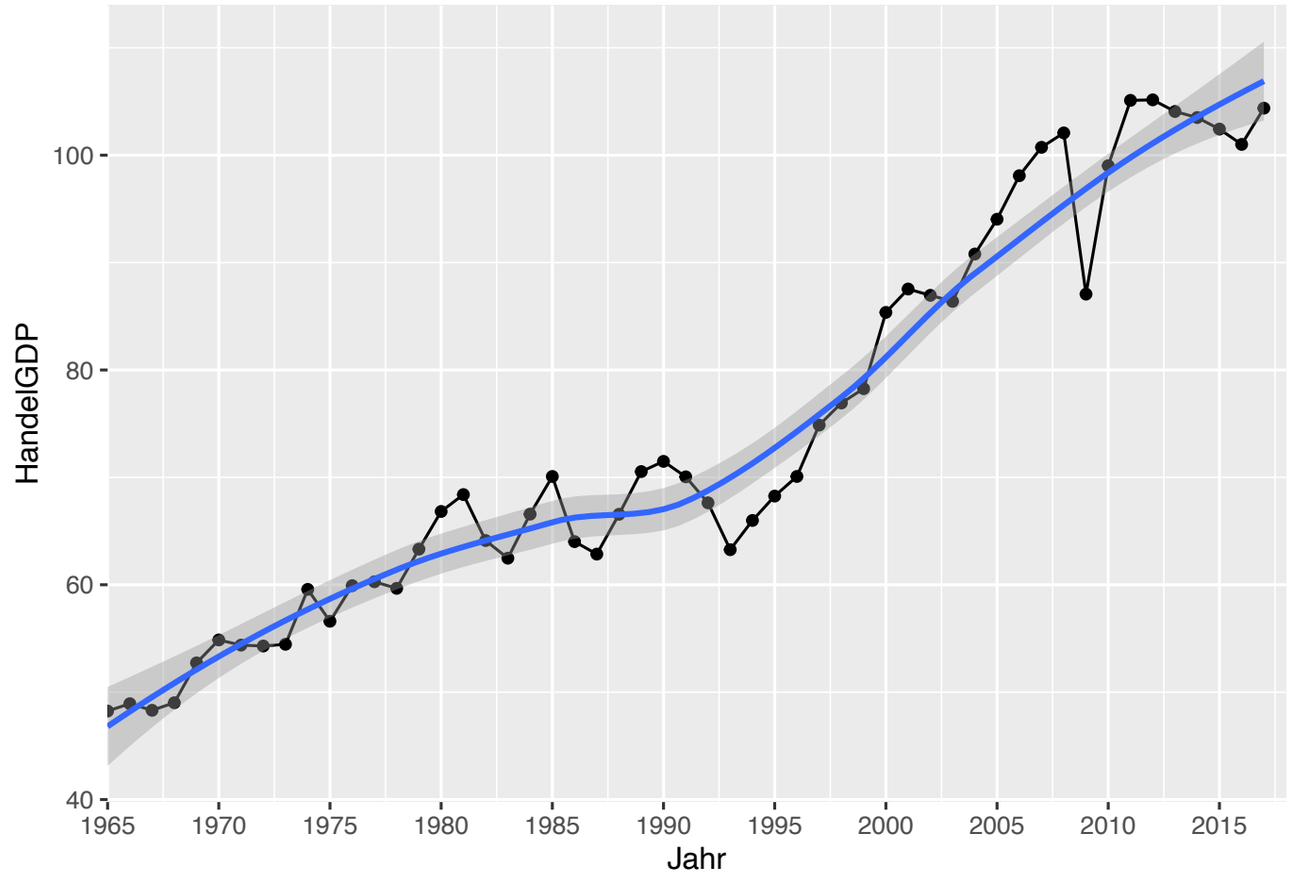


Datenvisualisierung mit ggplot2: ein Beispiel

- Nun wollen wir noch die Skalen anpassen:

```
aut_trade_plot <- aut_trade_plot +
  scale_x_continuous(limits = c(1965, 2018),
                     breaks = seq(1960, 2017, 5),
                     expand = c(0, 0)
                     )
aut_trade_plot
```

Zwischenergebnis:



Datenvisualisierung mit ggplot2: ein Beispiel

- Nun wollen wir noch die Skalen anpassen:

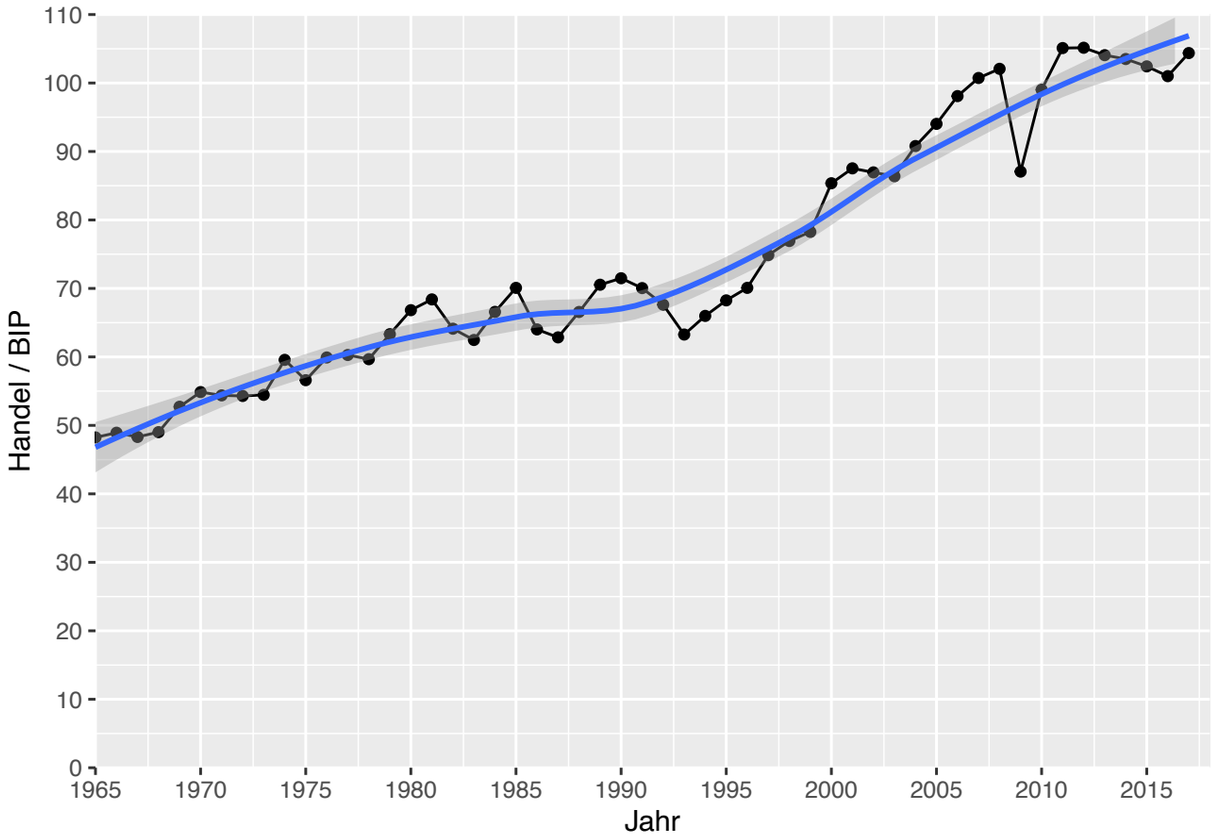
```
aut_trade_plot <- aut_trade_plot +
  scale_x_continuous(limits = c(1965, 2018),
                     breaks = seq(1960, 2017, 5),
                     expand = c(0, 0)
                     )

aut_trade_plot
```

- Für die y-Achse geht das äquivalent:

```
aut_trade_plot <- aut_trade_plot +
  scale_y_continuous(name = "Handel / BIP",
                     limits = c(0, 110),
                     breaks = seq(0, 110, 10),
                     expand = c(0, 0)
                     )
```

Zwischenergebnis:

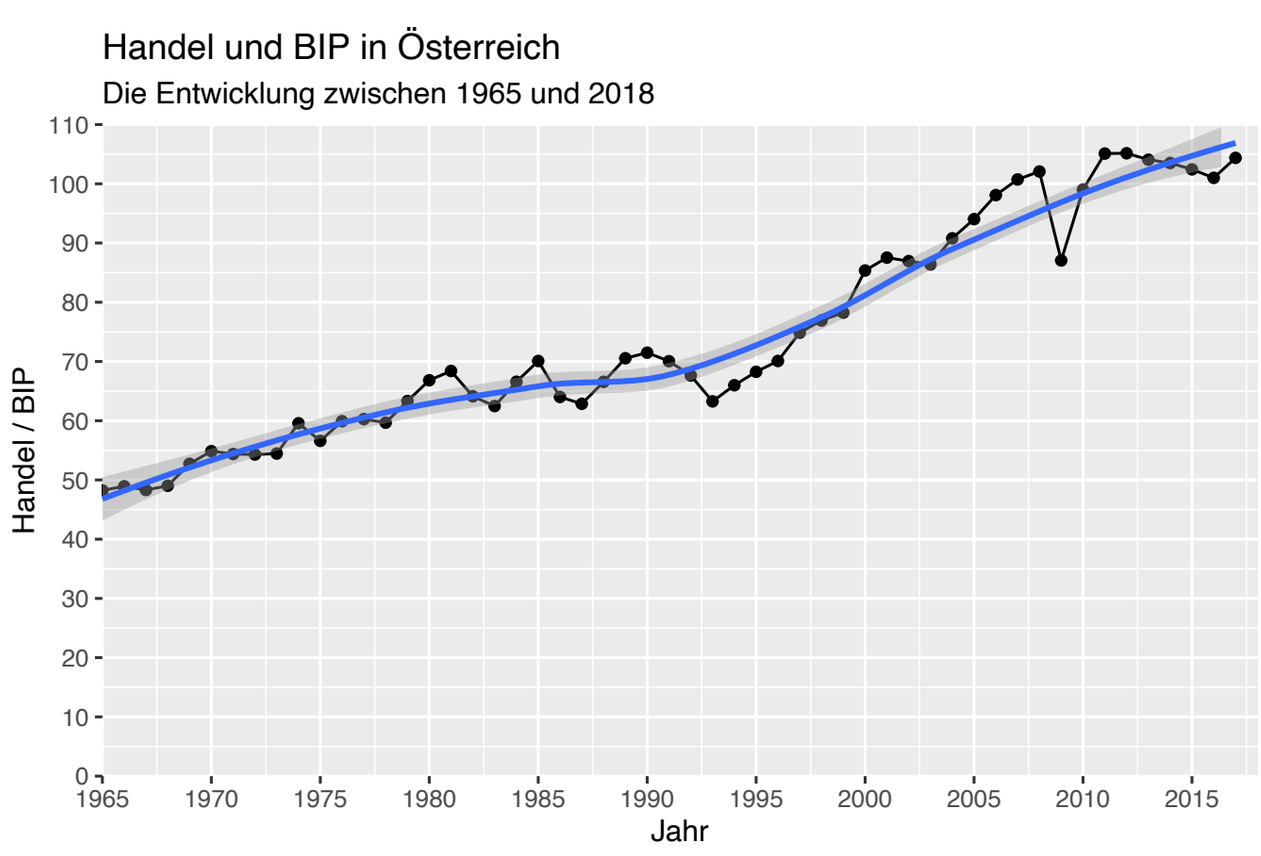


Datenvisualisierung mit ggplot2: ein Beispiel

- Jetzt fügen wir noch die *Labels* hinzu:

```
aut_trade_plot <- aut_trade_plot +
  labs(title = "Handel und BIP in Österreich",
           subtitle = "Die Entwicklung zwischen 1965 und 2018",
       caption = "Quelle: Weltbank.")
aut_trade_plot
```

Zwischenergebnis:

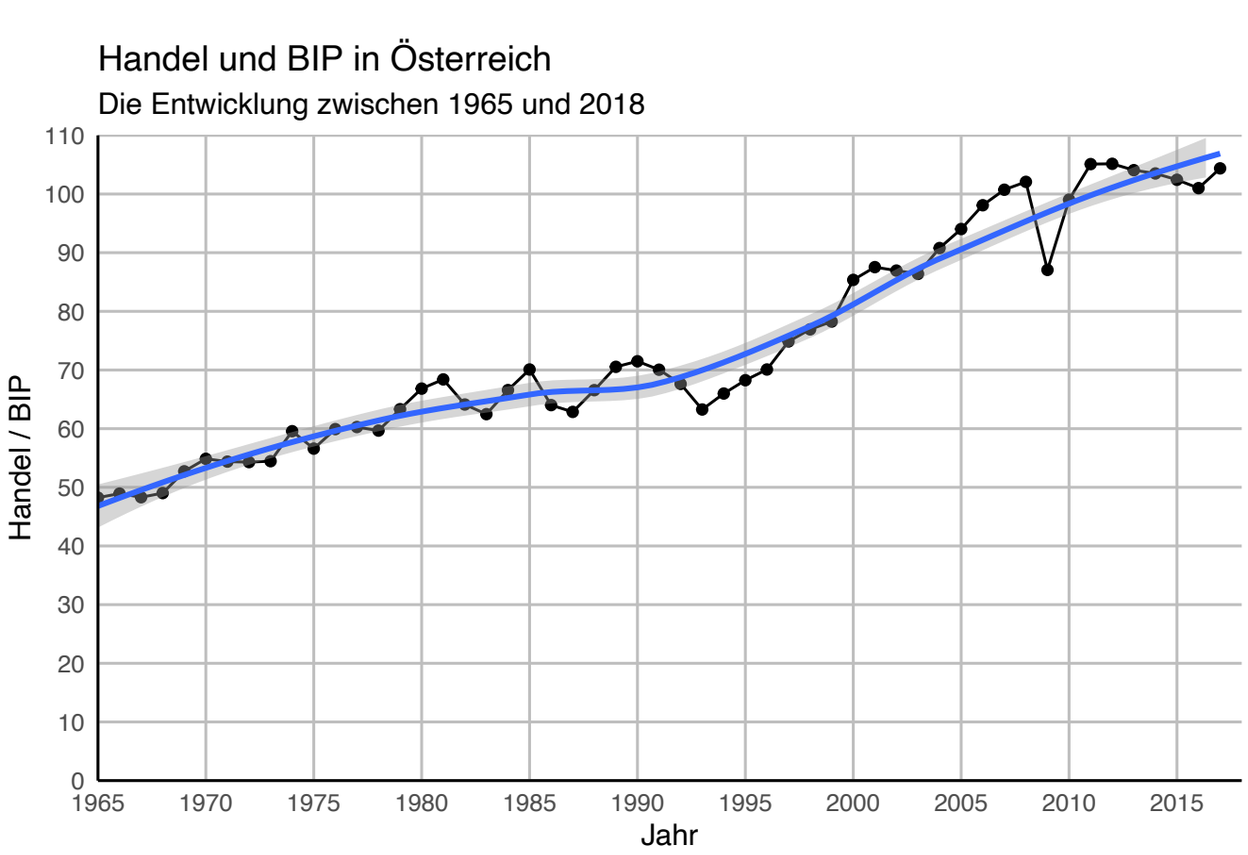


Datenvisualisierung mit ggplot2: ein Beispiel

- Alle restlichen Details können wir mit der Funktion **theme()** anpassen:

```
aut_trade_plot <- aut_trade_plot +
  theme(
    panel.background = element_rect(fill = "white"),
    panel.grid.major = element_line(colour = "grey"),
    panel.grid.minor = element_blank(),
    axis.line = element_line(colour = "black"),
    axis.ticks = element_blank()
    )
```

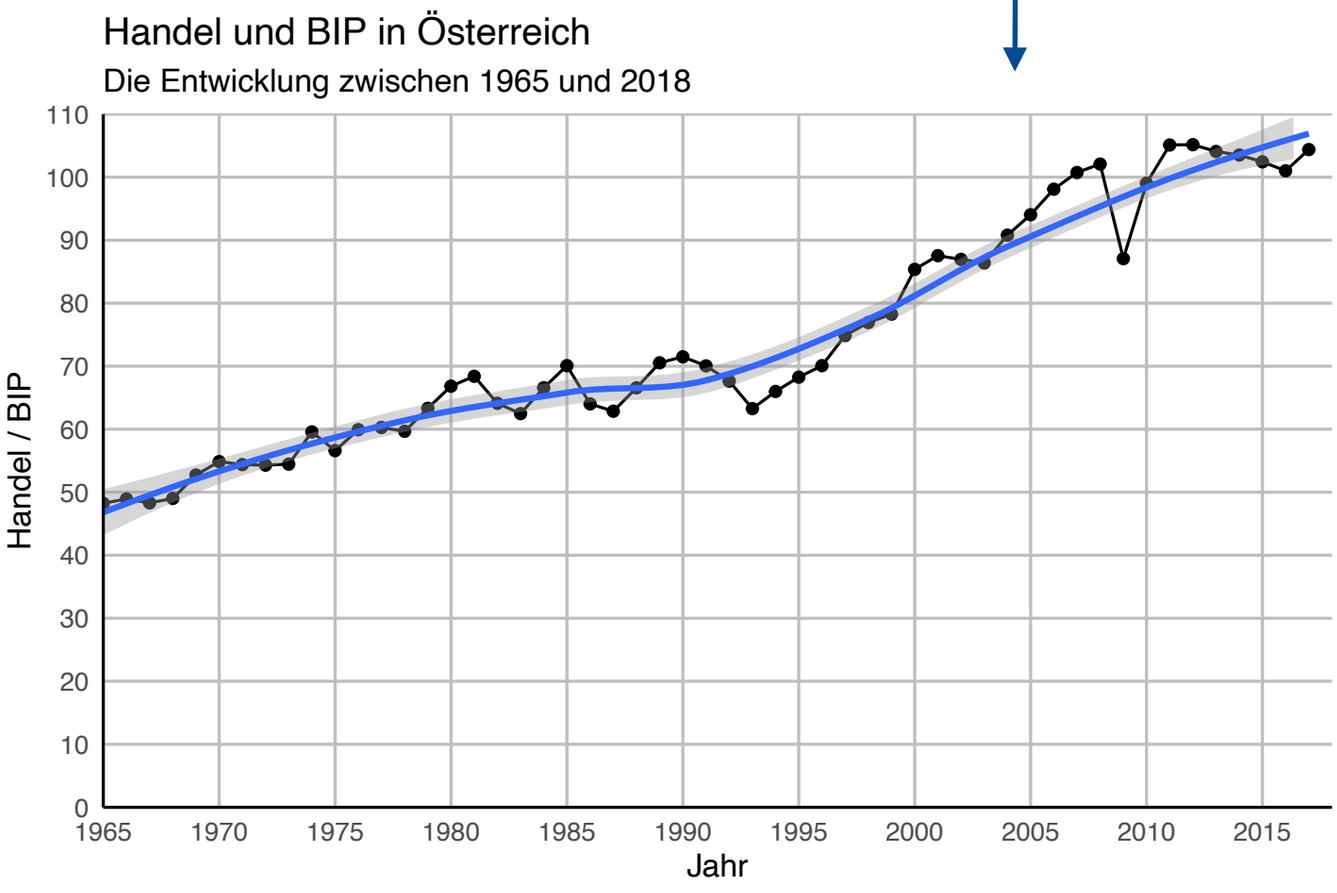
Zwischenergebnis:



Quelle: Weltbank.

Datenvisualisierung mit ggplot2: ein Beispiel

```
#>   Land Jahr HandelGDP
#> 1  AUT 1965  48.23931
#> 2  AUT 1966  48.92554
#> 3  AUT 1967  48.30854
#> 4  AUT 1968  49.01388
#> 5  AUT 1969  52.72526
#> 6  AUT 1970  54.86039
```




```
aut_trade_plot <- ggplot(
  data = aut_trade,
  mapping = aes(x = Jahr,
                y = HandelGDP)
) +
  geom_line() +
  geom_point() +
  geom_smooth() +
  scale_x_continuous(
    limits = c(1965, 2018),
    breaks = seq(1960, 2017, 5),
    expand = c(0, 0)
  ) +
scale_y_continuous(
  name = "Handel / BIP",
  limits = c(0, 110),
  breaks = seq(0, 110, 10),
  expand = c(0, 0)
) +
ggtitle(
  label = "Handel und BIP in Österreich",
  subtitle = "Die Entwicklung zwischen 1965 und 2018"
  ) +
theme(
  panel.background = element_rect(fill = "white"),
  panel.grid.major = element_line(colour = "grey"),
  panel.grid.minor = element_blank(),
  axis.line = element_line(colour = "black"),
  axis.ticks = element_blank()
)
```

Grafiken in ggplot2

- Das Prinzip ist bei allen Grafiken das gleiche
- Die Art der Grafik bestimmt sich nach den verwendeten *geoms*:

Art	**Anwendungsgebiet**	**Relevante Funktion**
Balkendiagramm	Vergleich von Werten	`geom_bar()`
Linienchart	Dynamiken	`geom_line()`, `geom_ribbon()`
Histogram	Verteilungen weniger Variablen	`geom_bar()`, `geom_hist()`, `geom_density()`
Streu- und Blasendiagramm	Zusammenhänge zwischen 2-4 variablen	`geom_point()`
Kuchendiagramm	Nichts	Keine

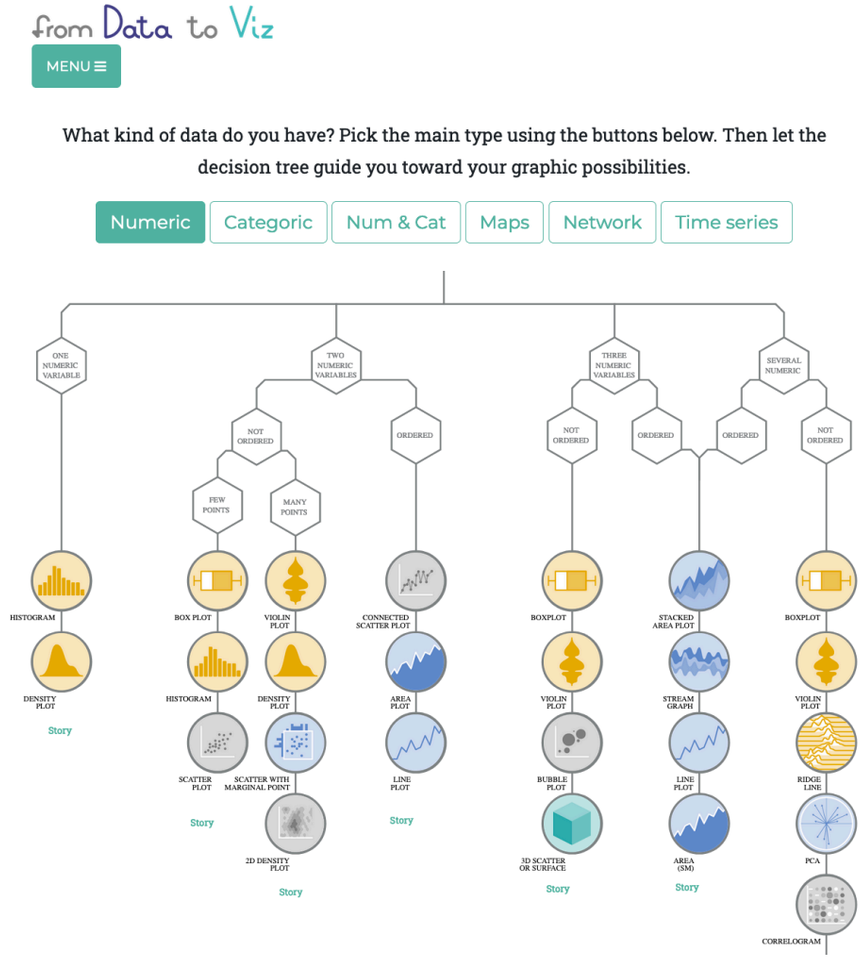
- Den Umgang lernt man am besten via *learning by doing*
- Zahlreiche Beispiele zum Nachbauen im Skript
  - Nutzen Sie die Diskussionsmöglichkeiten im Forum um sich gegenseitig zu helfen

Die Wahl der richtigen Grafik

- Verschiedene Visualisierungsformen passen zu unterschiedlichen Fragestellungen: https://www.data-to-viz.com/

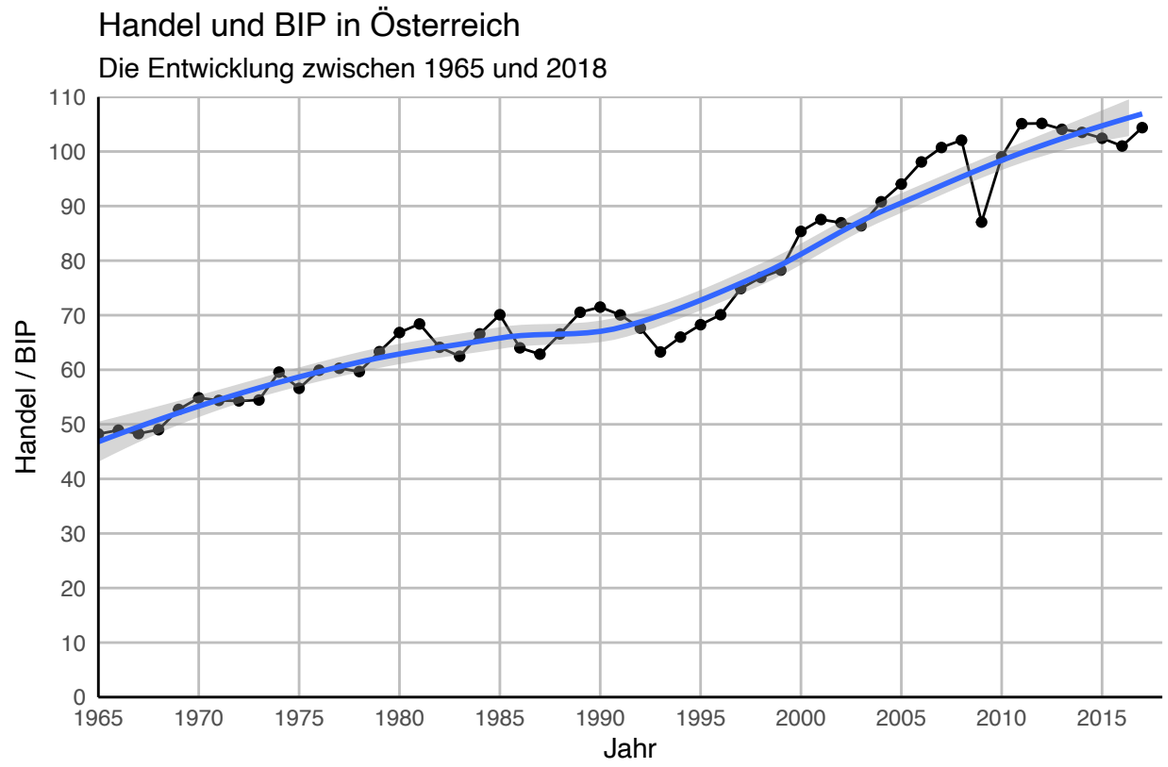
- Vier Grafiken, die viele Anwendungsfälle abdecken:

1. Der Linienchart
   - Besonders gut für die Darstellung von Dynamiken
2. Das Streudiagramm, bzw. der Bubble-Chart
   - Gut geeignet für Verhältnis von 2-3 Variablen
3. Der Box- oder Violinenplot
   - Gut geeignet für Gruppenvergleiche
4. Das Histogramm, bzw. die empirische Dichte
   - Vergleich verschiedener Variablen & Verteilungen



Der Linienchart

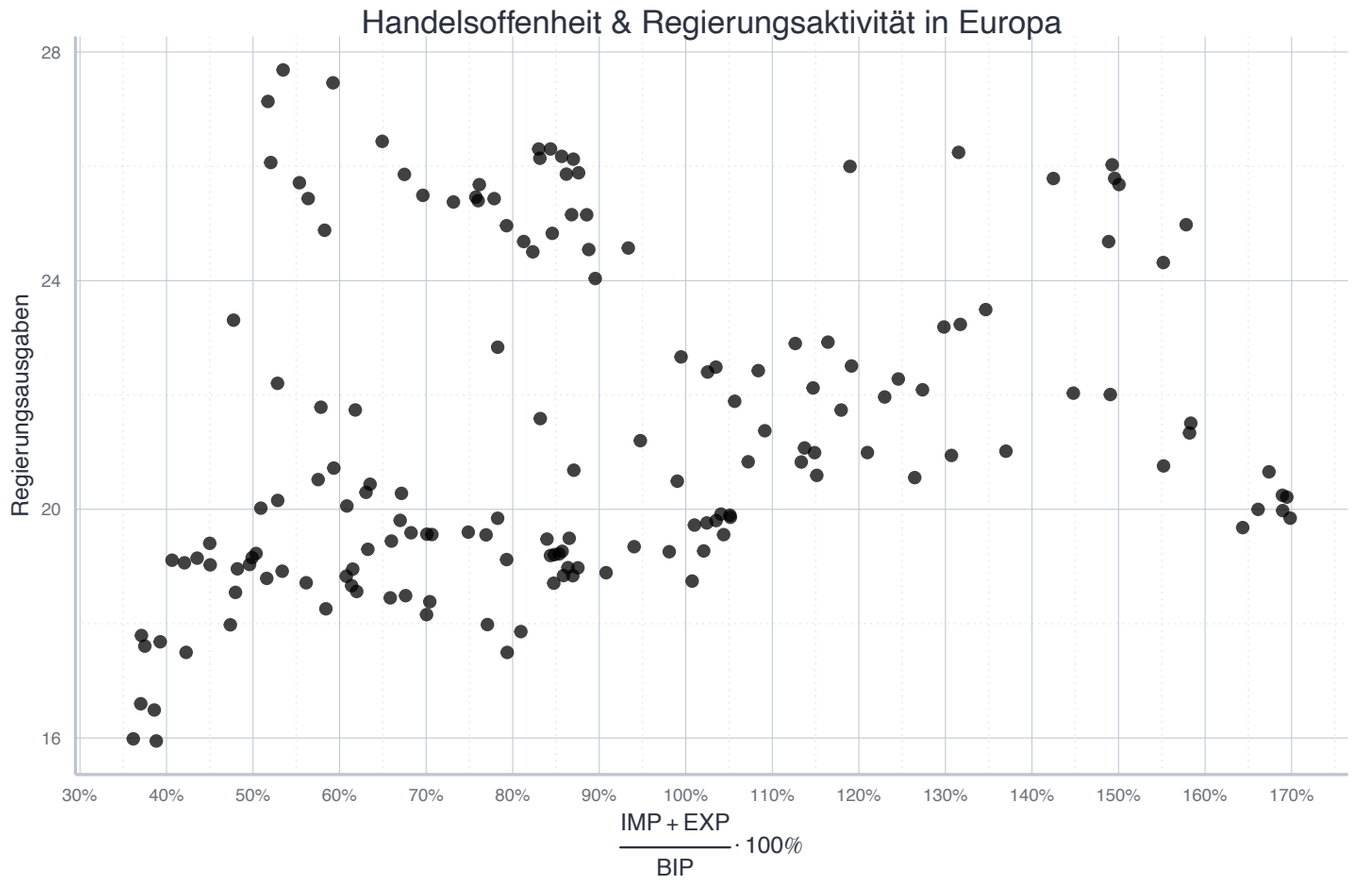
- Gut geeignet um zeitliche Entwicklungen darzustellen
- Häufig ist es hilfreich einen Trend oder eine Regressionsgerade hinzuzufügen
- Relevanter *geom*: **geom_line()**
- Nur mit wenigen Ästetiken (sinnvoll) kombinierbar:
  - color
  - linetype



Quelle: Weltbank.

Das Streudiagramm

- Gut geeignet wenn Sie das Verhältnis von 2-3 Variablen visualisieren wollen
- Die relevanten *geoms* sind hier Punkte: **geom_point()**



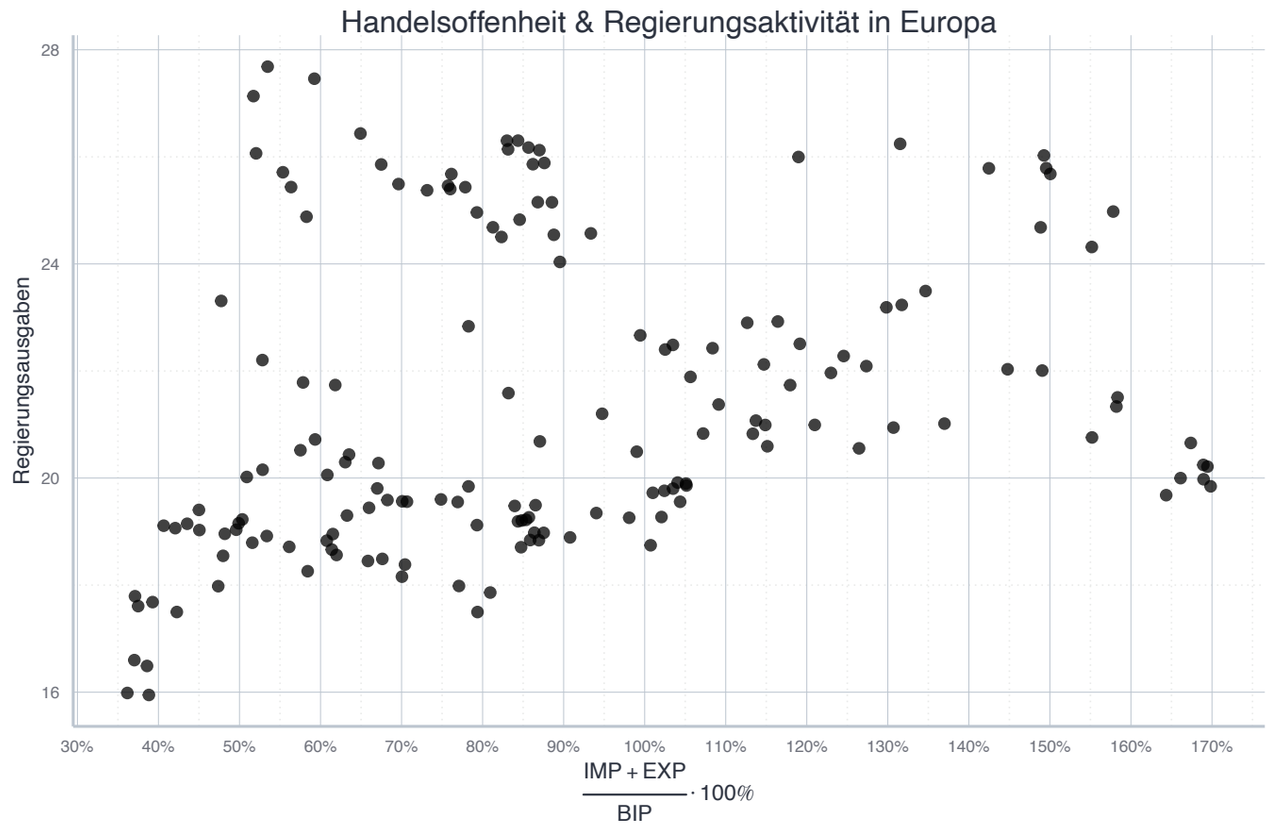
Quelle: Weltbank; Daten von 1990–2017.

Das Streudiagramm

```
head(offenheits_daten)
```

```
#>   year       Land trade_total_GDP gvnt_cons
#> 1 1991 Österreich        70.04841  18.15780
#> 2 1992 Österreich        67.63017  18.48991
#> 3 1993 Österreich        63.26505  19.30042
#> 4 1994 Österreich        65.98709  19.44437
#> 5 1995 Österreich        68.25660  19.58966
#> 6 1996 Österreich        70.08367  19.56574
```

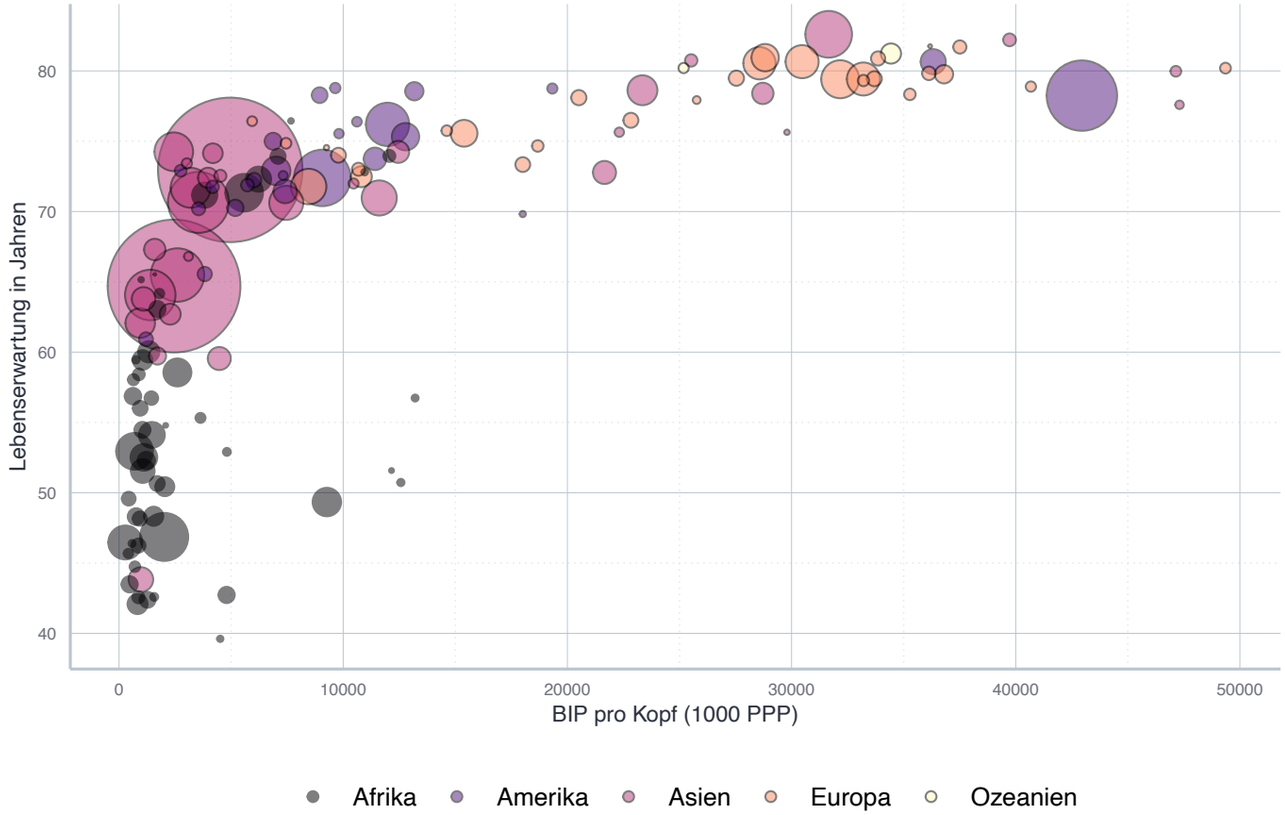
```
streudiagramm <- ggplot(
  data = offenheits_daten,
  mapping = aes(x=trade_total_GDP,
                y=gvnt_cons)
  ) +
  geom_point(alpha=0.75) +
  scale_y_continuous(name = "Regierungsausgaben") +
  scale_x_continuous(name = TeX("$\\frac{IMP + EXP}{BIP}\\cdot 100\\%$"),
                     breaks = seq(30, 180, 10),
                     labels = scales::percent_format(accuracy = 1, scale = 1)
                     )+
  labs(
    title = "Handelsoffenheit & Regierungsaktivität in Europa",
    caption = "Quelle: Weltbank; Daten von 1990-2017."
    ) +
  theme_icae()
streudiagramm
```



Quelle: Weltbank; Daten von 1990–2017.

Das Blasendiagramm

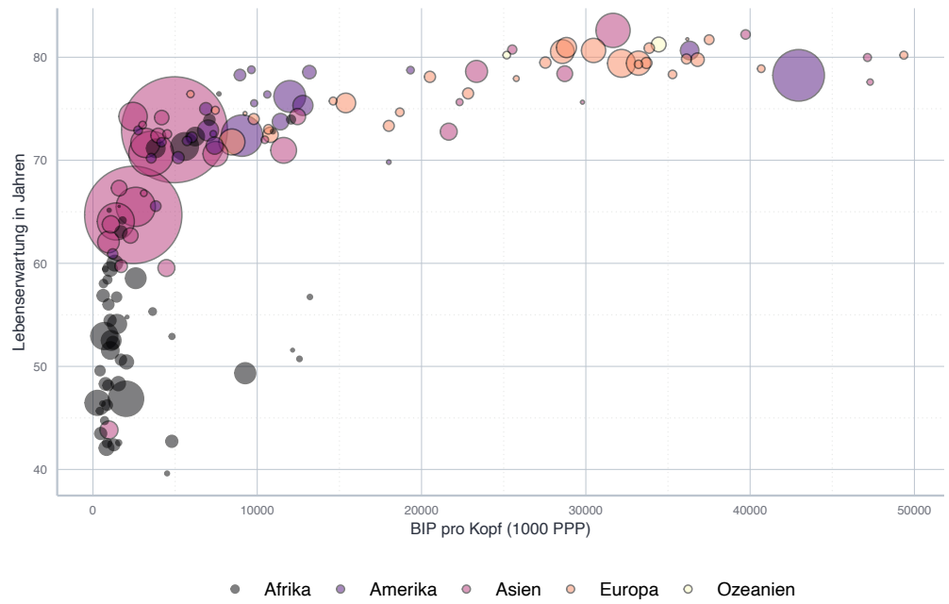
- Gut geeignet wenn Sie das Verhältnis von 2-3 Variablen visualisieren wollen
- Die relevanten *geoms* sind hier Punkte: geom_point()
- Bei bedarf können Sie die Punkte
  - ...nach Gruppen einfärben (color, fill)
  - ...nach einer dritten Variable skalieren (size)



Hinweis: Größe der Blasen repräsentiert Bevölkerungsanzahl. Quelle: Gapminder.

Das Blasendiagramm

```
bubble_plot <- ggplot(
  data = ausgangsdaten,
  mapping = aes(x = gdpPercap,
                y = lifeExp,
                size = pop,
                fill = continent)
    ) +
  geom_point(
    alpha=0.5, shape=21, color="black"
    ) +
  scale_size(
    range = c(0.1, 24), name="Bevölkerung", guide = FALSE
    ) +
  scale_fill_viridis(
    discrete=TRUE, option="A"
    ) +
  scale_y_continuous(
    name = "Lebenserwartung in Jahren"
    ) +
  scale_x_continuous(
    name = "BIP pro Kopf (1000 PPP)"
    ) +
  labs(
    caption = "Hinweis: Größe der Blasen repräsentiert Bevölkerungsanzahl. Quelle: Gapminder."
    ) +
  theme_icae() +
  theme(
    legend.position="bottom",
    plot.caption = element_text(hjust = 0)
    )
```



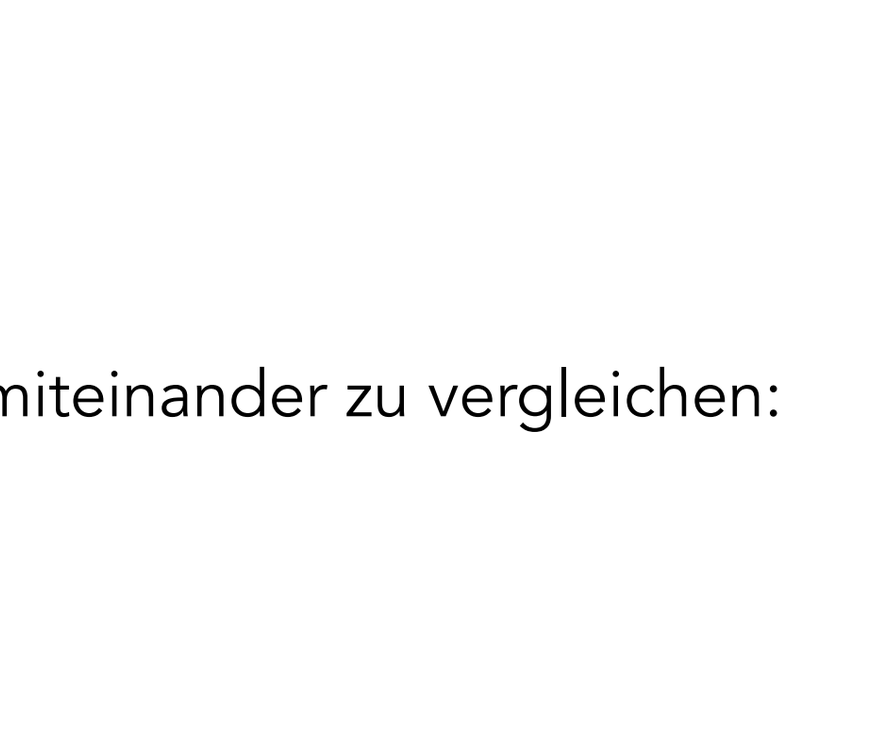
Hinweis: Größe der Blasen repräsentiert Bevölkerungsanzahl. Quelle: Gapminder.

Der Boxplot

- Der Boxplot wird häufig verwendet um Gruppen miteinander zu vergleichen:

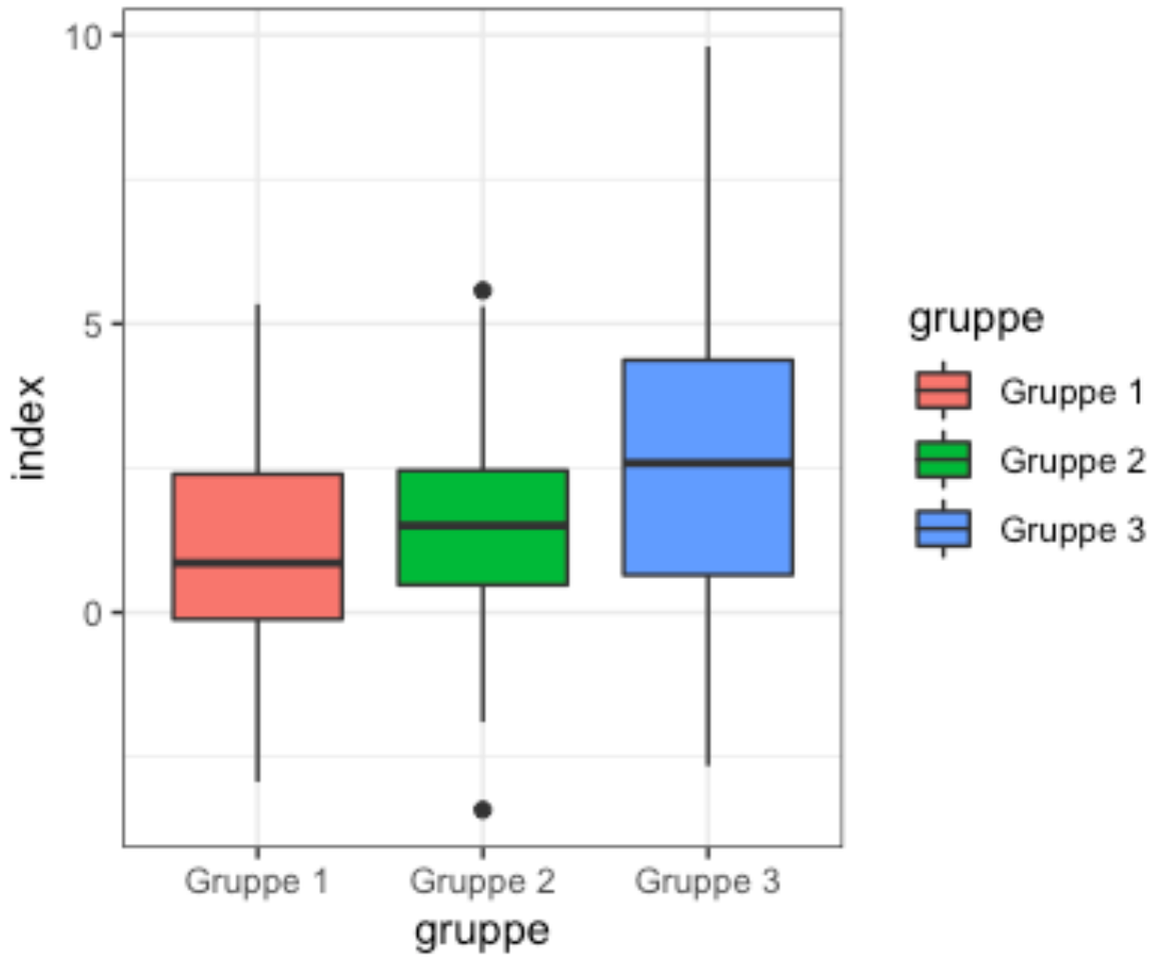
```
> datensatz
# A tibble: 150 x 3
      id  index gruppe
   <int>  <dbl> <chr>
 1     1 -0.121 Gruppe 1
 2     2  0.540 Gruppe 1
 3     3  4.12  Gruppe 1
 4     4  1.14  Gruppe 1
 5     5  1.26  Gruppe 1
 6     6  4.43  Gruppe 1
 7     7  1.92  Gruppe 1
 8     8 -1.53  Gruppe 1
 9     9 -0.374 Gruppe 1
10    10  0.109 Gruppe 1
# … with 140 more rows
```

```
ggplot(data = datensatz,
       mapping = aes(x=gruppe,
                     y=index,
                     fill=gruppe)
       ) +
  geom_boxplot() +
  theme_bw()
```



- Nicht schlecht, aber auch nicht schön!

Der Boxplot

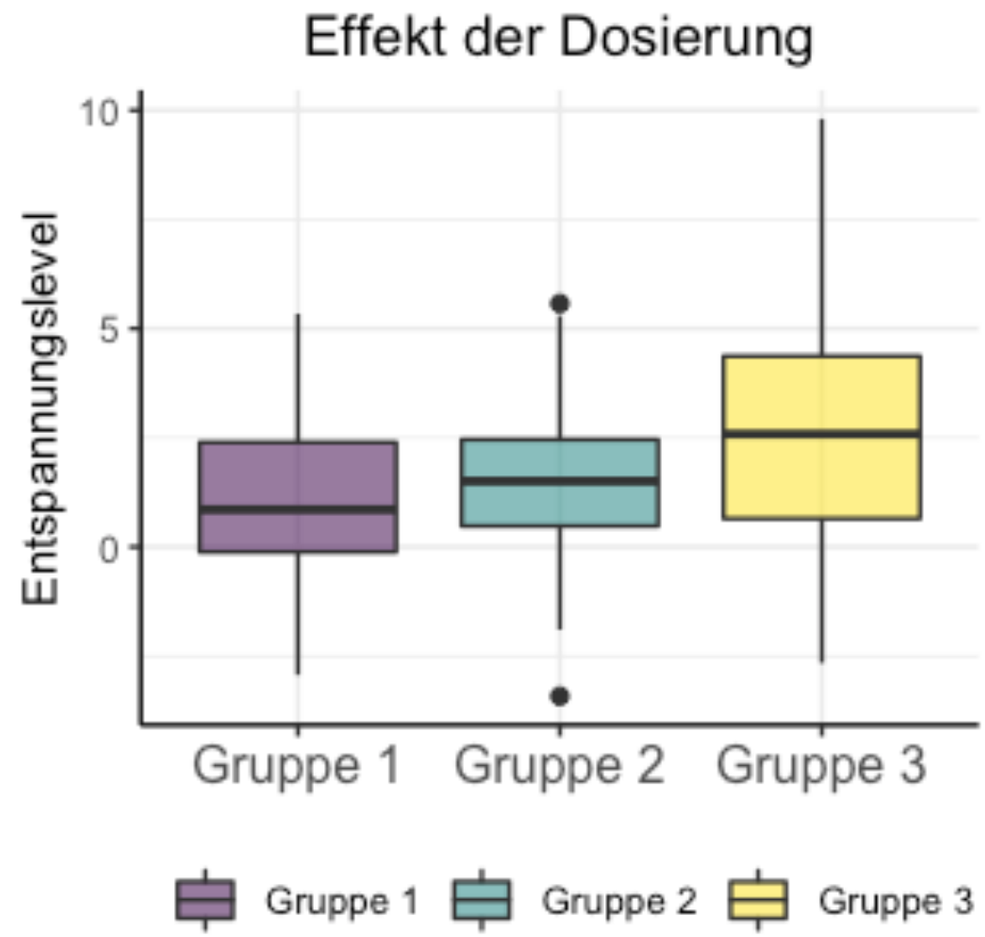


- Achsentitel komisch
- Kein Titel, keine Quelle
- Hässliches Farbschema
- Legende unschön
- Box um die Daten klobig
- …

Schritt-für-Schritt-Anpassung der Abbildung

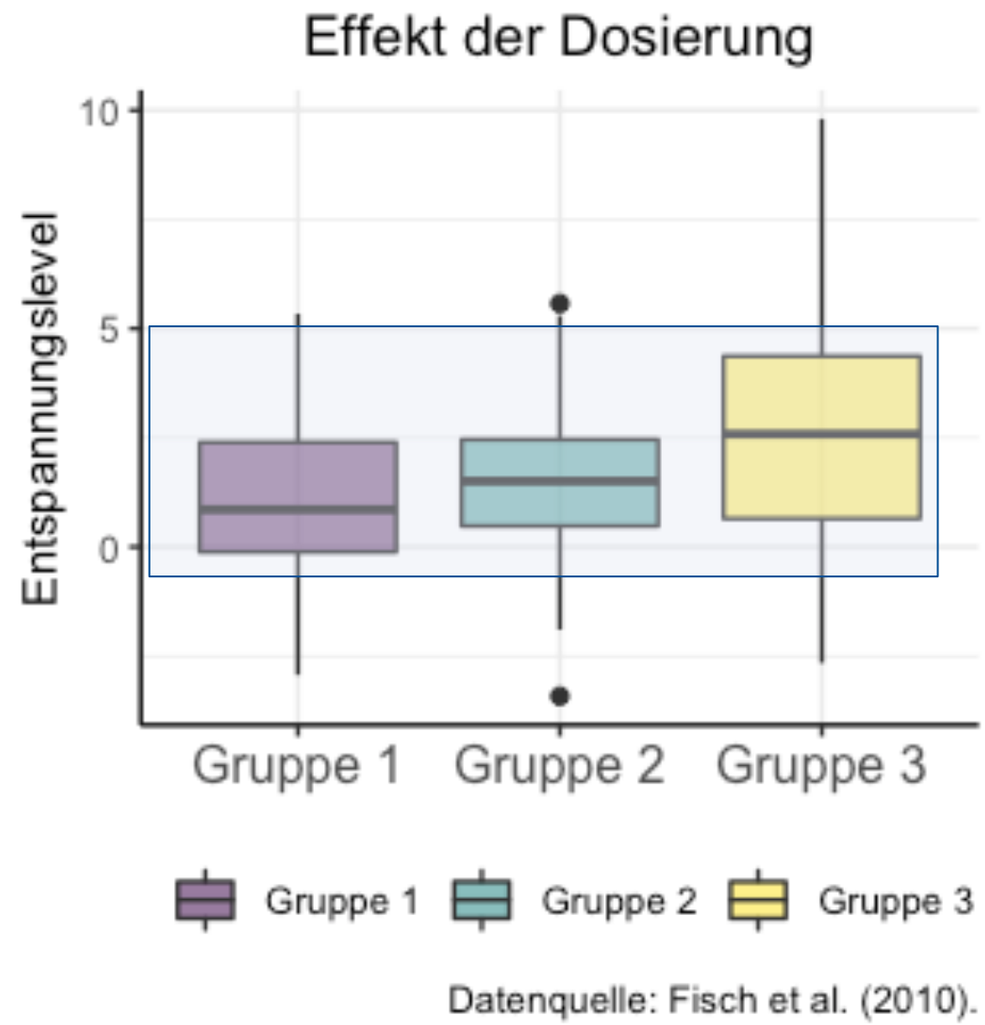
Der Boxplot

```
ggplot(data = datensatz,
       mapping = aes(x=gruppe,
                     y=index,
                     fill=gruppe)
) +
  geom_boxplot() +
  scale_fill_viridis(discrete = T, alpha = 0.6) +
  labs(title = "Effekt der Dosierung",
       caption = "Datenquelle: Fisch et al. (2010).",
       y = "Entspannungslevel") +
theme_bw() +
theme(
  plot.title = element_text(hjust=0.5),
  panel.border = element_blank(),
  axis.line = element_line(),
  axis.title.x = element_blank(),
  axis.text.x = element_text(size=12),
  legend.position = "bottom",
  legend.title = element_blank()
)
```



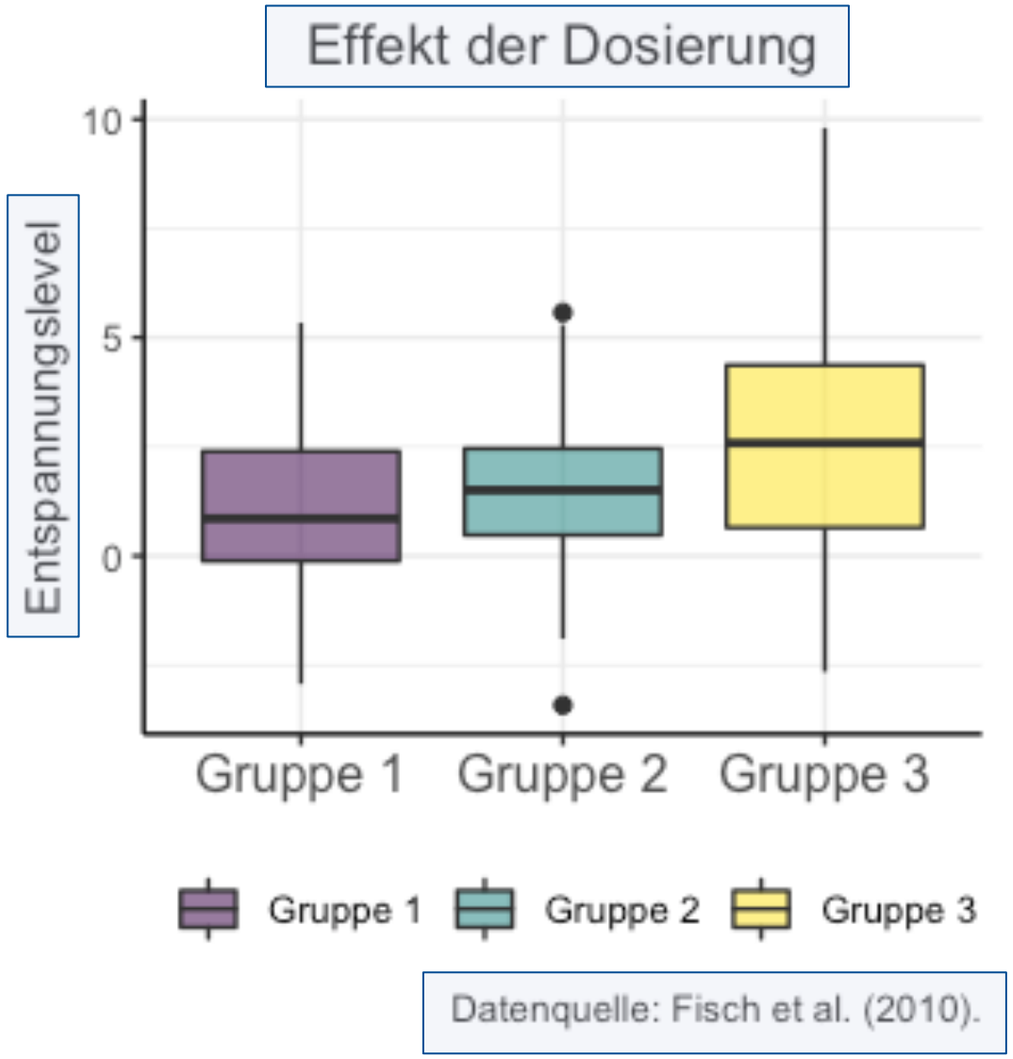
Der Boxplot

```
ggplot(data = datensatz,
       mapping = aes(x=gruppe,
                     y=index,
                     fill=gruppe)
) +
  geom_boxplot() +
  scale_fill_viridis(discrete = T, alpha = 0.6) +
  labs(title = "Effekt der Dosierung",
       caption = "Datenquelle: Fisch et al. (2010).",
       y = "Entspannungslevel") +
theme_bw() +
theme(
  plot.title = element_text(hjust=0.5),
  panel.border = element_blank(),
  axis.line = element_line(),
  axis.title.x = element_blank(),
  axis.text.x = element_text(size=12),
  legend.position = "bottom",
  legend.title = element_blank()
)
```



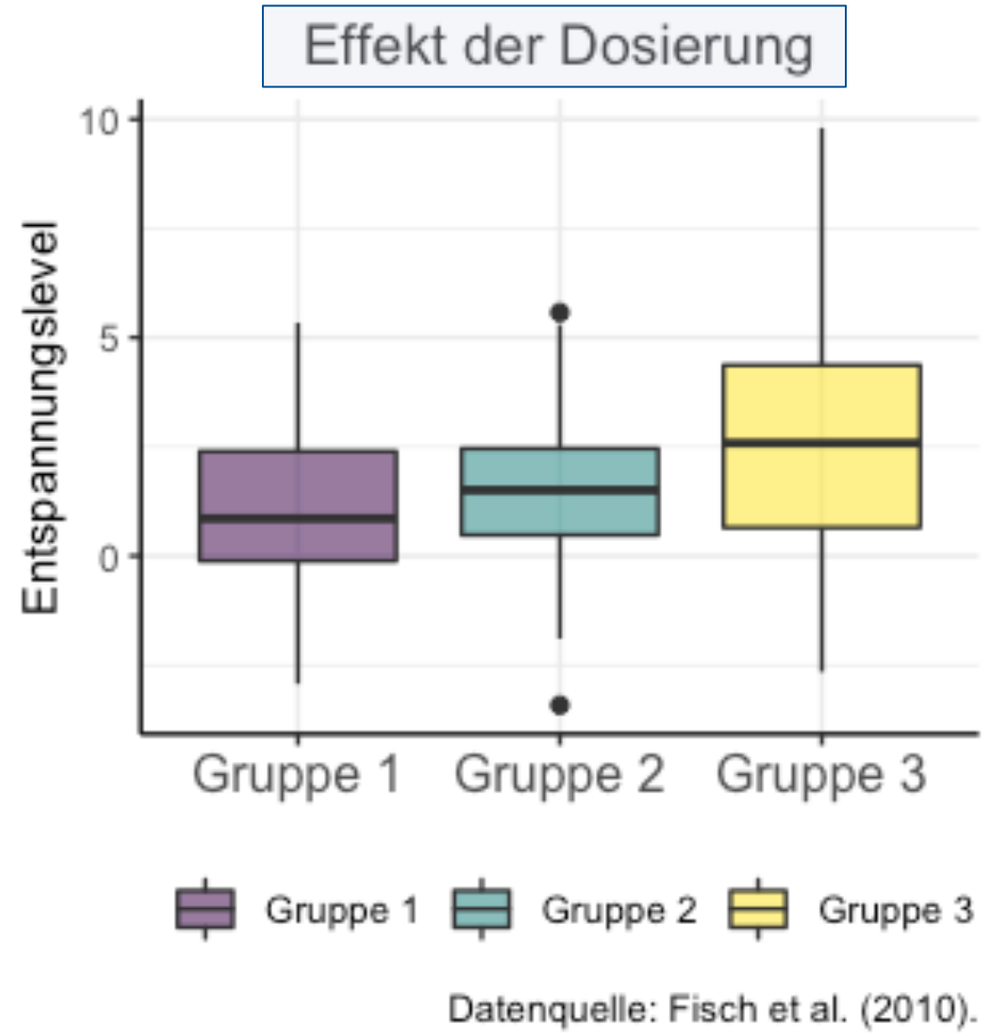
Der Boxplot

```
ggplot(data = datensatz,
       mapping = aes(x=gruppe,
                     y=index,
                     fill=gruppe)
) +
  geom_boxplot() +
  scale_fill_viridis(discrete = T, alpha = 0.6) +
  labs(title = "Effekt der Dosierung",
       caption = "Datenquelle: Fisch et al. (2010).",
       y = "Entspannungslevel") +
theme_bw() +
theme(
  plot.title = element_text(hjust=0.5),
  panel.border = element_blank(),
  axis.line = element_line(),
  axis.title.x = element_blank(),
  axis.text.x = element_text(size=12),
  legend.position = "bottom",
  legend.title = element_blank()
)
```



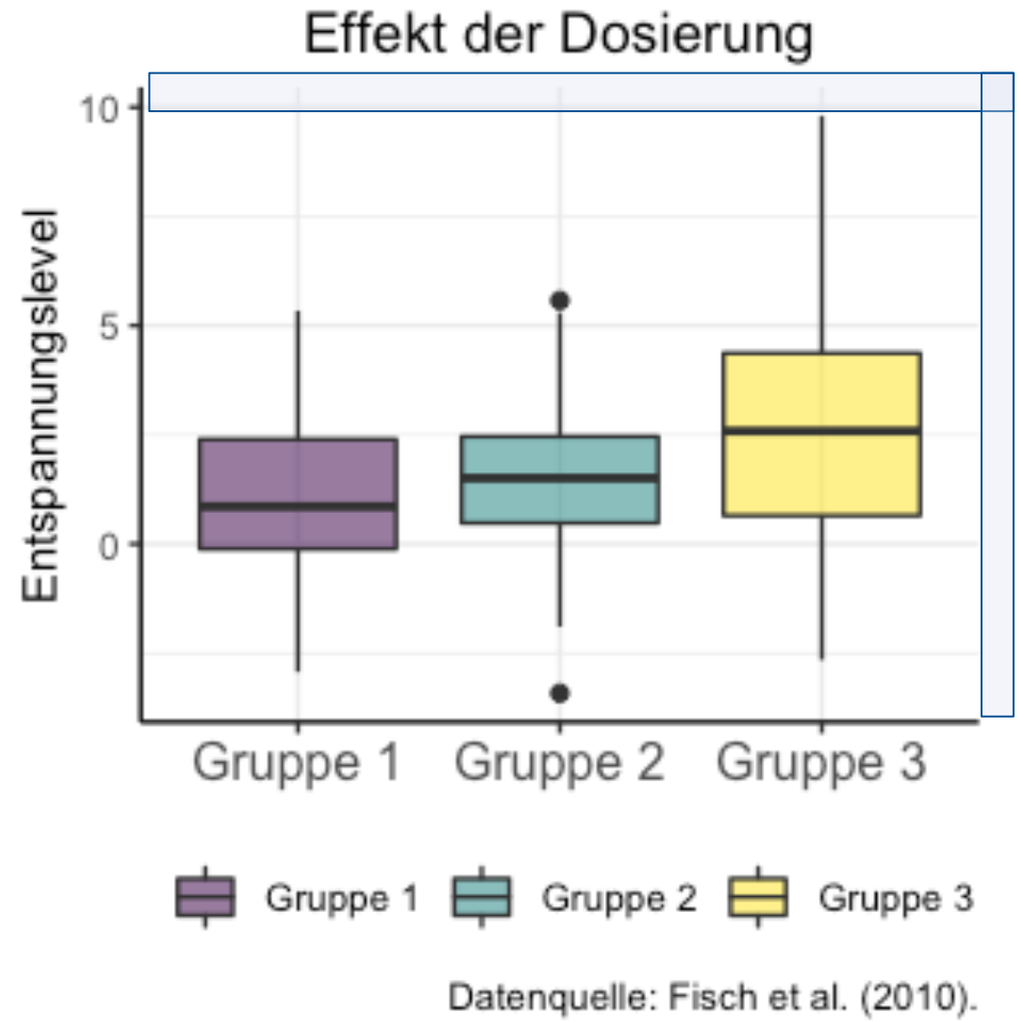
Der Boxplot

```
ggplot(data = datensatz,
       mapping = aes(x=gruppe,
                     y=index,
                     fill=gruppe)
) +
  geom_boxplot() +
  scale_fill_viridis(discrete = T, alpha = 0.6) +
  labs(title = "Effekt der Dosierung",
       caption = "Datenquelle: Fisch et al. (2010).",
       y = "Entspannungslevel") +
theme_bw() +
theme(
  plot.title = element_text(hjust=0.5),
  panel.border = element_blank(),
  axis.line = element_line(),
  axis.title.x = element_blank(),
  axis.text.x = element_text(size=12),
  legend.position = "bottom",
  legend.title = element_blank()
)
```



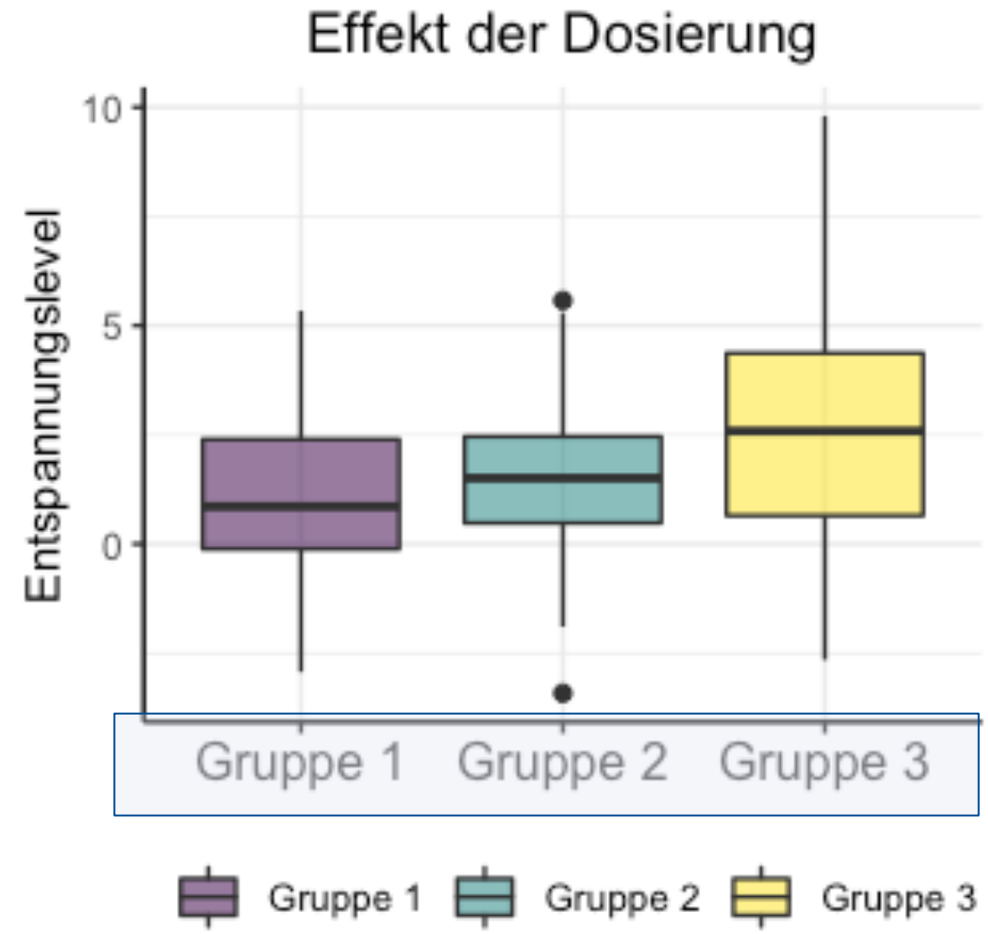
Der Boxplot

```
ggplot(data = datensatz,
       mapping = aes(x=gruppe,
                     y=index,
                     fill=gruppe)
) +
  geom_boxplot() +
  scale_fill_viridis(discrete = T, alpha = 0.6) +
  labs(title = "Effekt der Dosierung",
       caption = "Datenquelle: Fisch et al. (2010).",
       y = "Entspannungslevel") +
theme_bw() +
theme(
  plot.title = element_text(hjust=0.5),
  panel.border = element_blank(),
  axis.line = element_line(),
  axis.title.x = element_blank(),
  axis.text.x = element_text(size=12),
  legend.position = "bottom",
  legend.title = element_blank()
)
```



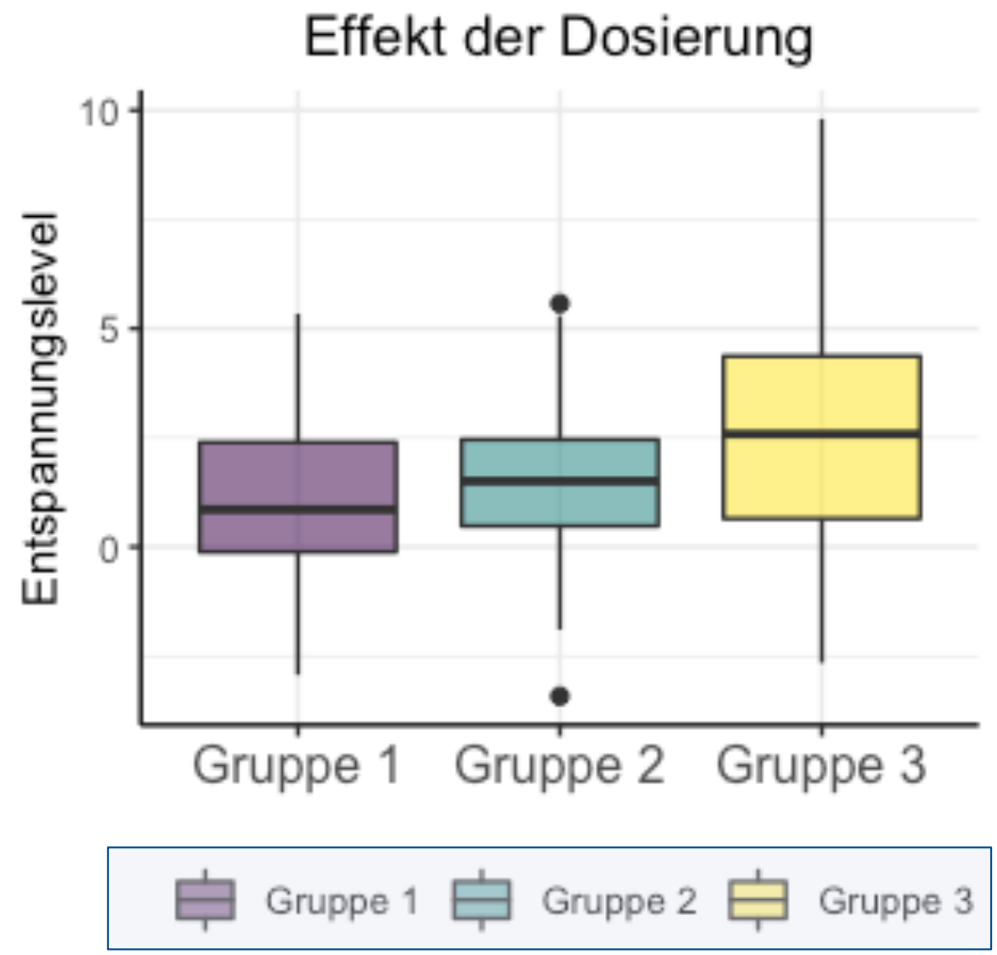
Der Boxplot

```
ggplot(data = datensatz,
       mapping = aes(x=gruppe,
                     y=index,
                     fill=gruppe)
) +
  geom_boxplot() +
  scale_fill_viridis(discrete = T, alpha = 0.6) +
  labs(title = "Effekt der Dosierung",
       caption = "Datenquelle: Fisch et al. (2010).",
       y = "Entspannungslevel") +
theme_bw() +
theme(
  plot.title = element_text(hjust=0.5),
  panel.border = element_blank(),
  axis.line = element_line(),
  axis.title.x = element_blank(),
  axis.text.x = element_text(size=12),
  legend.position = "bottom",
  legend.title = element_blank()
)
```



Der Boxplot

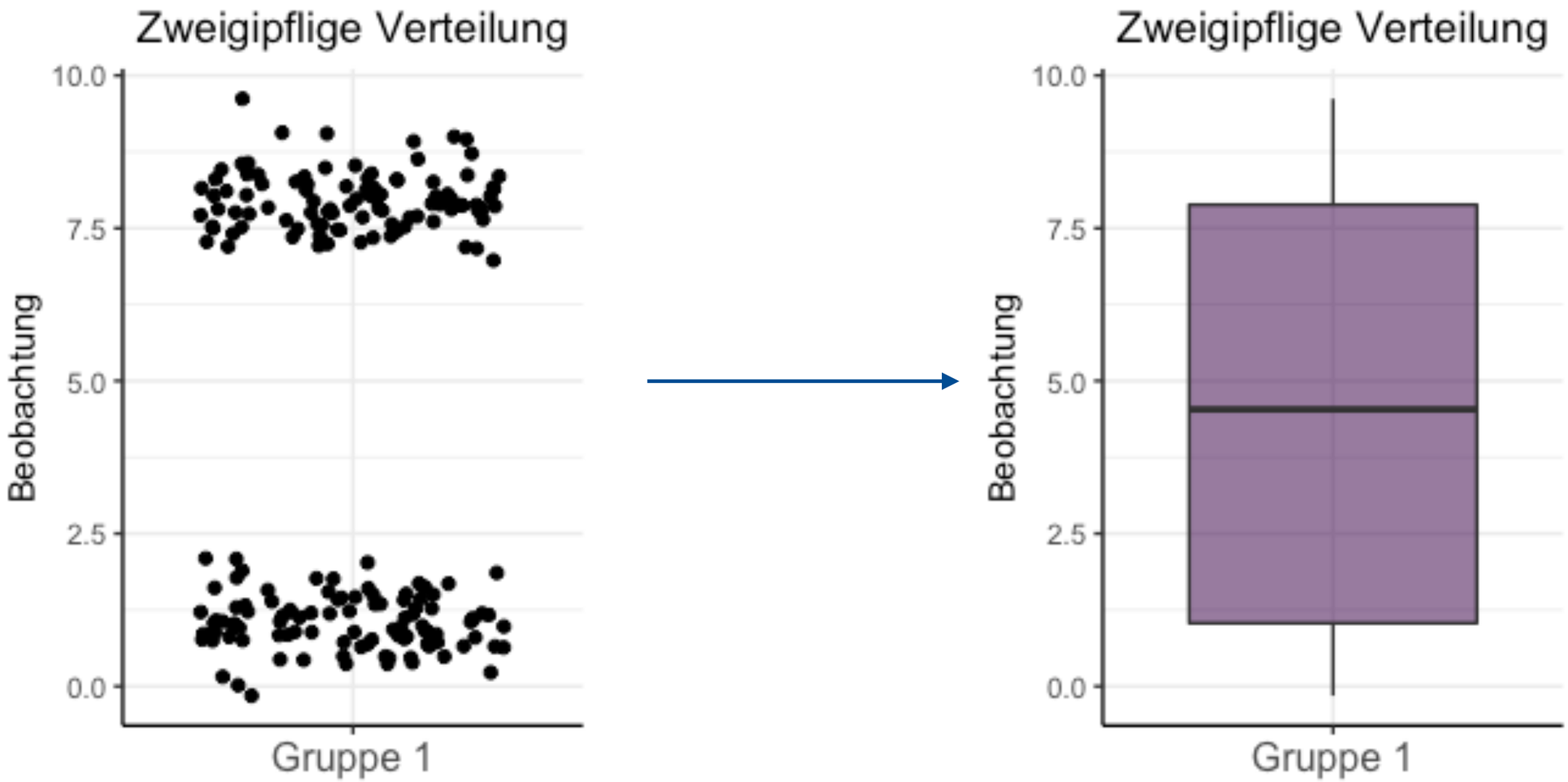
```
ggplot(data = datensatz,
       mapping = aes(x=gruppe,
                     y=index,
                     fill=gruppe)
) +
  geom_boxplot() +
  scale_fill_viridis(discrete = T, alpha = 0.6) +
  labs(title = "Effekt der Dosierung",
       caption = "Datenquelle: Fisch et al. (2010).",
       y = "Entspannungslevel") +
theme_bw() +
theme(
  plot.title = element_text(hjust=0.5),
  panel.border = element_blank(),
  axis.line = element_line(),
  axis.title.x = element_blank(),
  axis.text.x = element_text(size=12),
  legend.position = "bottom",
  legend.title = element_blank()
)
```



Der Boxplot

Mögliche Probleme und Alternativen

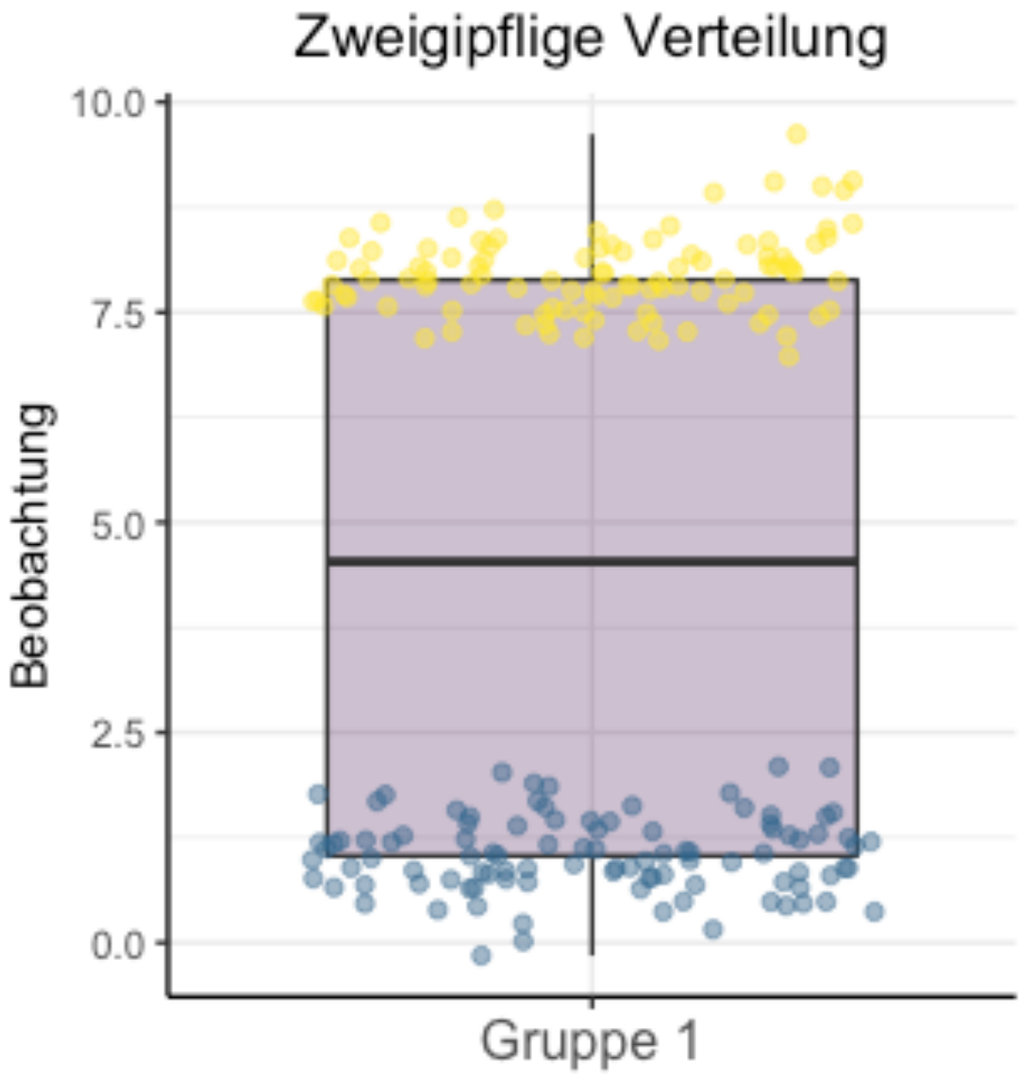
- Der Boxplot verdeckt potenzielle Besonderheiten der Verteilung
  - Zweigipfligkeit zum Beispiel nicht erkennbar:



Der Boxplot

Mögliche Probleme und Alternativen

- Lösung 1: mit geom_jitter() die Beobachtungen ergänzen
- Lösung 2: einen Violinenplot verwenden

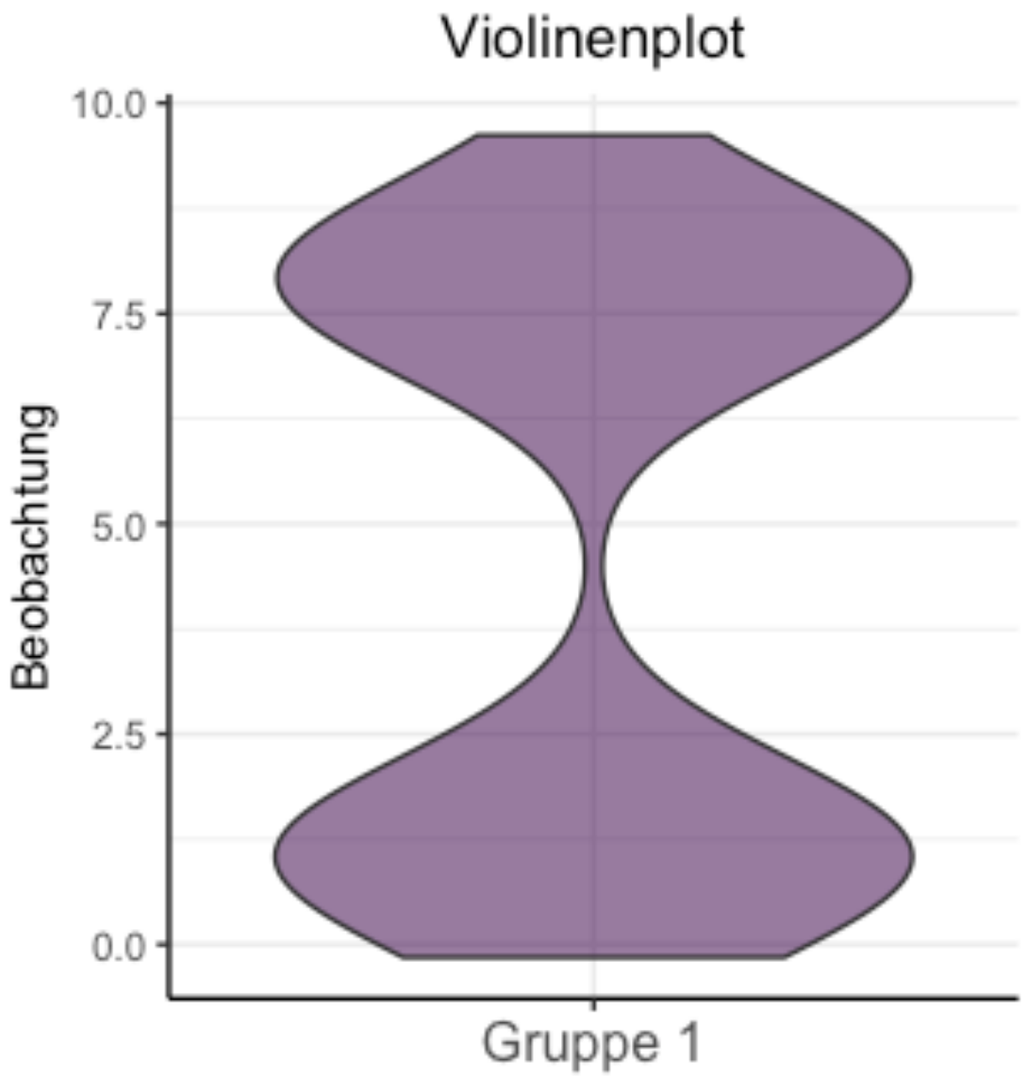



```
ggplot(data = zweigipfel_daten,
       mapping = aes(x=Gruppe,
                     y=Beobachtung,
                     fill=Gruppe)
) +
  geom_boxplot() +
  geom_jitter(alpha=0.5, aes(color=Sub_Gruppe)) +
  scale_fill_viridis(discrete = T, alpha = 0.3) +
  scale_color_viridis(discrete = T, begin = 0.33,
                      end = 1) +
  theme_bw() +
  labs(title="Zweigipflige Verteilung") +
  theme(
    plot.title = element_text(hjust=0.5),
    panel.border = element_blank(),
    axis.line = element_line(),
    axis.title.x = element_blank(),
    axis.text.x = element_text(size=12),
    legend.position = "none",
    legend.title = element_blank()
  )
```

Der Boxplot

Mögliche Probleme und Alternativen

- Lösung 1: mit geom_jitter() die Beobachtungen ergänzen
- Lösung 2: einen Violinenplot verwenden



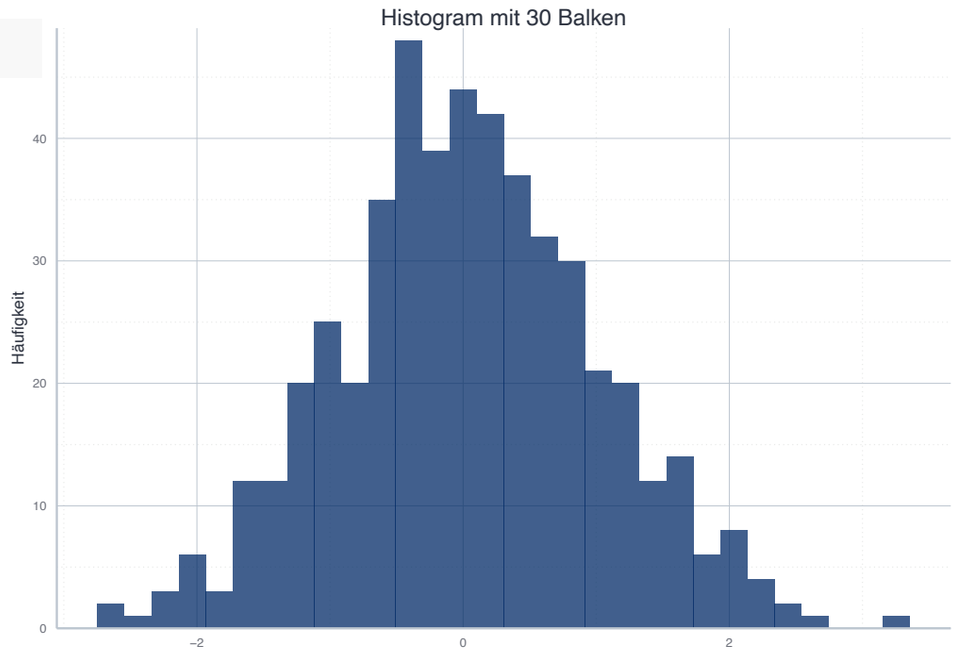

```
ggplot(data = zweigipfel_daten,
       mapping = aes(x=Gruppe,
                     y=Beobachtung,
                     fill=Gruppe)
) +
  geom_violin() +
  scale_fill_viridis(discrete = T, alpha = 0.6) +
  theme_bw() +
  labs(title="Violinenplot") +
  theme(
    plot.title = element_text(hjust=0.5),
    panel.border = element_blank(),
    axis.line = element_line(),
    axis.title.x = element_blank(),
    axis.text.x = element_text(size=12),
    legend.position = "none",
    legend.title = element_blank()
  )
```

Das Histogramm bzw. die Dichtefunktion

- Das Histogramm ist eine gute Wahl um die Verteilung einer Variable zu zeigen
- Relevante geom: geom_histogram()
- Zentrale Herausforderung: Richtige Wahl der 'Blöcke'

```
head(histogram_daten)
```

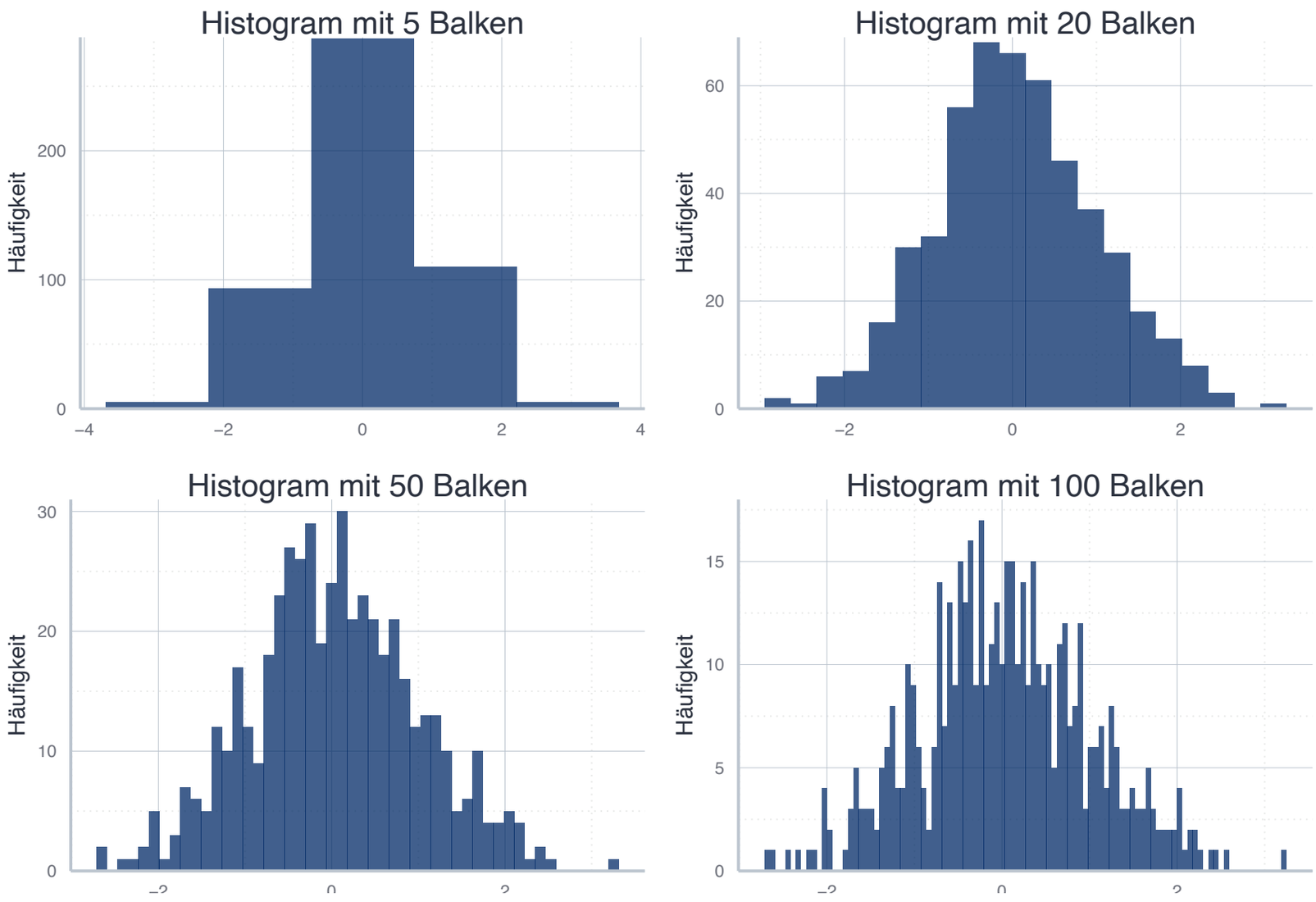
```
#>            x
#> 1 -0.56047565
#> 2 -0.23017749
#> 3  1.55870831
#> 4  0.07050839
#> 5  0.12928774
#> 6  1.71506499
```




```
ggplot(data = histogram_daten,
       mapping = aes(x=x)) +
  geom_histogram(alpha=0.75, color=NA, fill="#002966") +
  scale_y_continuous(name = "Häufigkeit",
                     expand = expand_scale(c(0, 0), c(0, 1))) +
  ggtitle("Histogram mit 30 Balken") +
  theme_icae() +
  theme(axis.title.x = element_blank())
```

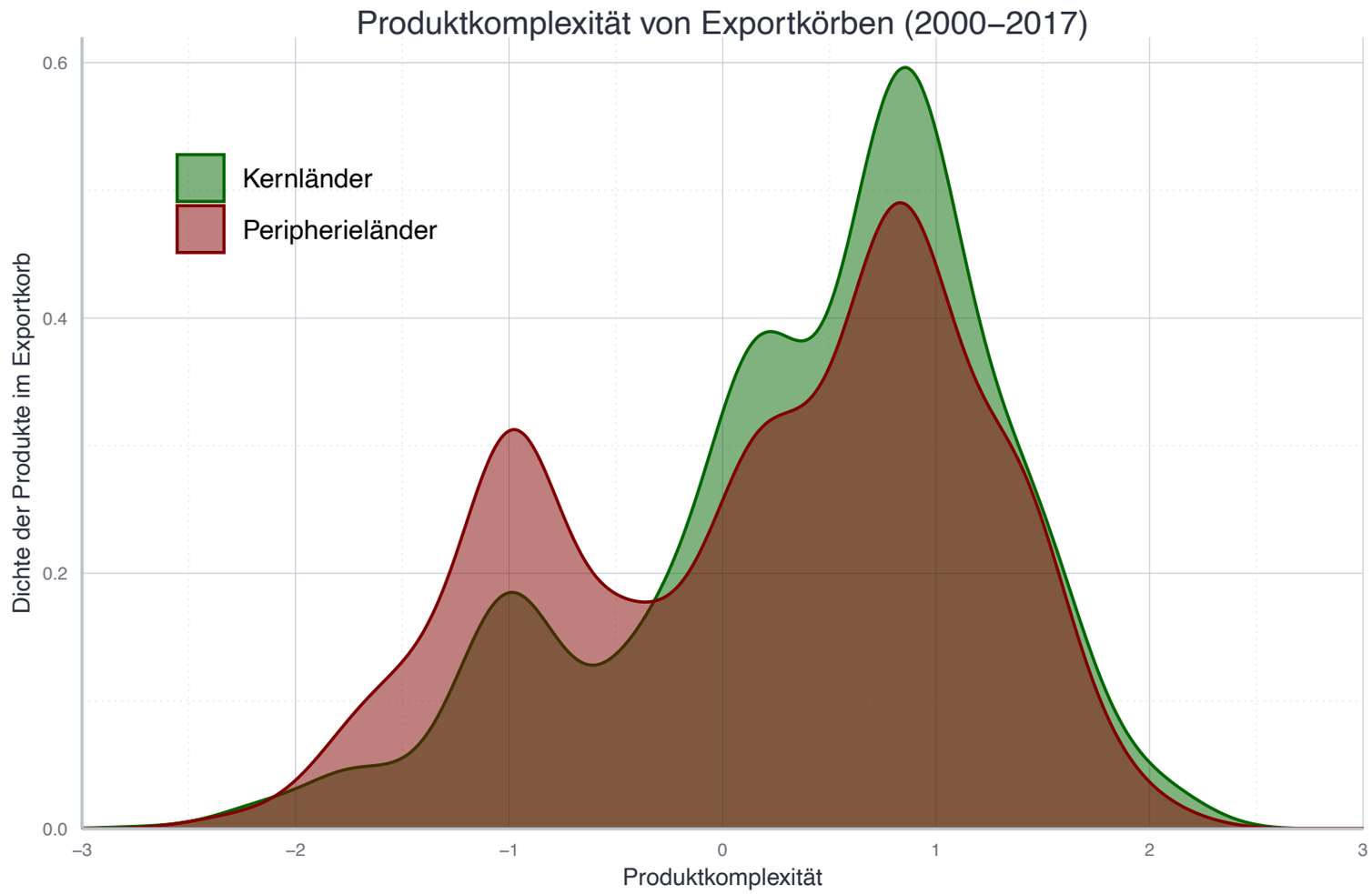
Das Histogramm bzw. die Dichtefunktion

- Es gibt einige *Heuristiken* nach denen man die Anzahl der Blöcke wählt
- Am Ende des Tages aber eine pragmatische Entscheidung i.S.d. Message, die Sie vermitteln wollen



Das Histogramm bzw. die Dichtefunktion

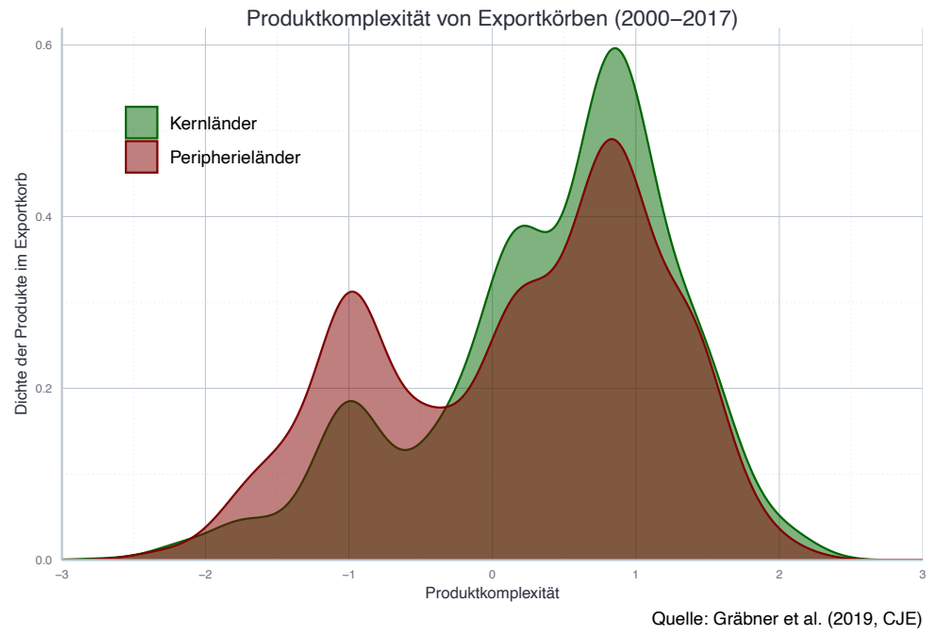
- Bei sehr vielen Beobachtungen können Sie um das Problem der Blockanzahl umgehen indem Sie eine Dichte für die Daten schätzen
- Relevantes *geom*: geom_density()
- Kann auch mit color und fill kombiniert werden



Quelle: Gräbner et al. (2019, CJE)

Das Histogramm bzw. die Dichtefunktion

```
ggplot(data = exportzusammensetzung,
       mapping = aes(
         x=pci,
         color=cgroup,
         fill=cgroup)
       ) +
 geom_density(
   mapping = aes(weight=exp_share),
   alpha=0.5
   ) +
 labs(
   title = "Produktkomplexität von Exportkörben (2000-2017)",
   caption = "Quelle: Gräbner et al. (2019, CJE)"
  ) +
ylab("Dichte der Produkte im Exportkorb") +
xlab("Produktkomplexität") +
scale_y_continuous(limits = c(0, 0.62), expand = c(0, 0)) +
scale_x_continuous(limits = c(-3, 3), expand = c(0, 0)) +
scale_color_icae(palette = "mixed", aesthetics = c("color", "fill")) +
theme_icae() +
theme(legend.position = c(0.175, 0.8))
```



Zusammenfassung

- Datenakquise und -aufbereitung als potenziell zeitaufwändige aber unvermeidliche Arbeitsschritte
  - Es lohnt sich die relevanten Algorithmen gut zu beherrschen
- Zentrale Anforderung: alle Ergebnisse müssen aus den Rohdaten heraus reproduzierbar sein
- Der Datenvisualisierung in ggplot2 sind wenig Grenzen gesetzt
  - Grafiken werden Stück für Stück beschrieben und durch den Aufruf der beschreibenden Liste gerendert
  - Am besten schauen Sie sich die Beispiele im Skript an
- Details zu allen verwendeten Funktionen finden Sie im Skript
  - Notwendig/hilfreich für die Aufgabenblätter
  - Zudem: im Skript ausführliche Erläuterung zum Thema “Manipulieren mit Grafiken”

Wiederholungsfragen

- Unter welchen Umständen kann man eine empirische Forschungsarbeit als ‘empirisch und inklusiv’ bezeichnen?
- Wie sollte die Ordnerstruktur in einem quantitativen Forschungsprojekt aussehen?
- Welche vier Skalenniveaus von Daten können wir unterscheiden? Wie werden die Daten jeweils in R repräsentiert?
- Worin genau unterscheiden sich intervall- und verhältnisskalierte Daten?

Wiederholungsfragen

- *Hinweis: für diese Vorlesung ist die beste Wiederholung wohl das eigenständige Nachprogrammieren der Aufbereitungsschritte.*
- Was verstehen wir unter “tidy data”?
- Was unterscheidet breite von langen Datensätzen und wo liegen jeweils die Anwendungsgebiete?
- Warum ist es im Zweifel besser die Spaltentypen beim Einlesen von Daten manuell über colClasses direkt anzugeben?
- Welche Varianten der *_join()-Familie haben Sie kennen gelernt und wann werden sie jeweils verwendet?

Wiederholungsfragen

- Was für einen Datentyp produziert die ggplot()-Funktion?
- Wofür werden die geom_*()-Funktionen verwendet?
- Mit welcher Funktion können sie in einem kleinen Text unten links die Quelle ihrer Daten gut angeben?
- Skizzieren Sie kurz die Anwendungsgebiete der folgenden Abbildungstypen:
  - Liniendiagramm
  - Streudiagramm
  - Blasendiagramm
  - Histogramm
  - Dichtefunktion