

# 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)

Wintersemester 2021/22

Letzte Aktualisierung: 20. Oktober 2021

## Einführung in das Tidyverse

# Inhalte und Ziele der Sitzung

- Erweiterbarkeit von R mit Paketen
- Moderne Syntax für die Datenanalyse
- Verständnis der häufigsten Schritte bei der Datenmanipulation
- Verständnis **ordentlicher** Datenstrukturen



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

- Leseempfehlung: [Tidyverse Cookbook \(Golemund, 2020\)](#)

# Pakete



# Pakete I

- Die Basisfunktionalität von R wird durch Pakete erweitert.
- Pakete enthalten Funktionen und/oder Objekte (z. B. Datensätze).
- Pakete in R kann man sich wie einen Werkzeugkasten vorstellen.
- Bevor man ein Werkzeug nutzen kann, muss man den Werkzeugkasten jedoch erst **ein Mal** anschaffen.



`install.packages()`

»Werkzeugkasten kaufen«

→ **Ein Mal**



`library()`

»Werkzeug benutzen«

→ **Zu Beginn jeder neuen R-Sitzung**

- Pakete werden mit der Funktion `install.packages()` installiert. Der Paketname steht in Anführungszeichen, z. B.:

```
install.packages("beispieldpaket")
```

- Um die Funktionen aus dem Paket **beispieldpaket** in der Datenanalyse zu nutzen, muss dieses Paket nach der einmaligen Installation geladen werden.
- Pakete werden mit der Funktion `library()` geladen. Hier muss der Paketname nicht in Anführungsstrichen stehen, z. B.:

```
library(beispieldpaket)
```

- Die in der Datenanalyse benötigten Pakete werden gesammelt zu Beginn des R-Skripts geladen.

# Das Tidyverse

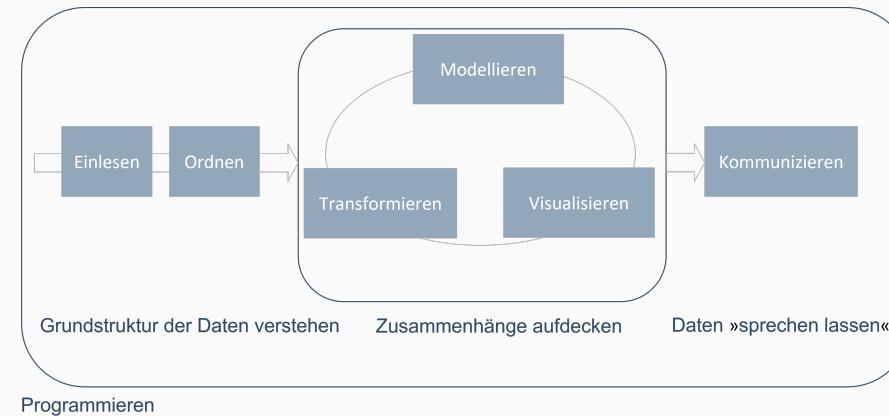


# Was ist das Tidyverse?

- Die Autoren des Tidyverse beschreiben es so:

Das tidyverse ist eine Sammlung von R-Paketen, die für die Data Science entwickelt wurden. Alle Pakete teilen eine zugrunde liegende Design-Philosophie, Grammatik und Datenstrukturen.

- Für jeden der klassischen Schritte eines Datenanalyseprojekts gibt es nützliche Funktionen aus dem Tidyverse.



- Das Tidyverse legt großen Wert auf eine gute (Menschen-)Lesbarkeit des Codes.
- Das macht Coden intuitiv und leichter zugänglich.

# Grundlegendes Prinzip: ordentliche Daten (*tidy data*)

Die drei Bedingungen für ordentliche Daten:

country	year	cases	population
Afghanistan	2000	45	197071
Afghanistan	2000	566	2095360
Brazil	1999	3737	17206362
Brazil	2000	8188	17404898
China	1999	21358	12725572
China	2000	21666	12806583

Variablen

Jede Variable in einer eigenen Spalte.

country	year	cases	population
Afghanistan	1999	45	197071
Afghanistan	2000	566	2095360
Brazil	1999	3737	17206362
Brazil	2000	8188	17404898
China	1999	21358	12725572
China	2000	21666	12806583

Beobachtungen

Jede Beobachtung in einer eigenen Zeile.

country	year	cases	population
Afghanistan	1999	45	197071
Afghanistan	2000	566	2095360
Brazil	1999	3737	17206362
Brazil	2000	8188	17404898
China	1999	21358	12725572
China	2000	21666	12806583

Werte

Jeder Wert in einer eigenen Zelle.

# Grundlegendes Prinzip: ordentliche Daten (*tidy data*)

- **Jede Variable in einer eigenen Spalte.**
- Jede Beobachtung in einer eigenen Zeile.
- Jeder Wert in einer eigenen Zelle.

Beispiel **dirty**:

land_a	land_b
1000000	1100000

## Was stimmt hier nicht?

- Hier stehen die beiden Länder in getrennten Spalten.
- Der Wert in den Zellen beschreibt das BIP.

## Lösung

- In der Variable (Spalte) `land` werden die Namen der Länder aufgelistet.
- In der Variable `bip` werden die zugehörigen Werte der BIPs eingetragen.

Beispiel **tidy**:

land	bip
a	1000000
b	1100000

# Grundlegendes Prinzip: ordentliche Daten (*tidy data*)

- Jede Variable in einer eigenen Spalte.
- **Jede Beobachtung in einer eigenen Zeile.**
- Jeder Wert in einer eigenen Zelle.

Beispiel **dirty**:

land	bip
a, b	1000000, 1100000

## Was stimmt hier nicht?

- Sowohl bei der Variable `bip` als auch `land` sind zwei Werte eingetragen.

## Lösung

- Jeder Beobachtung (hier: `land`) wird eine eigene Zeile zugewiesen.

Beispiel **tidy**:

land	bip
a	1000000
b	1100000

# Grundlegendes Prinzip: ordentliche Daten (*tidy data*)

- Jede Variable in einer eigenen Spalte.
- Jede Beobachtung in einer eigenen Zeile.
- **Jeder Wert in einer eigenen Zelle.**

## Was stimmt hier nicht?

- Die Variable `bip/einwohner` enthält zwei Werte, die durch `/` getrennt sind.

Beispiel **dirty**:

land	bip/einwohner
a	1000000/5000
b	1100000/6250

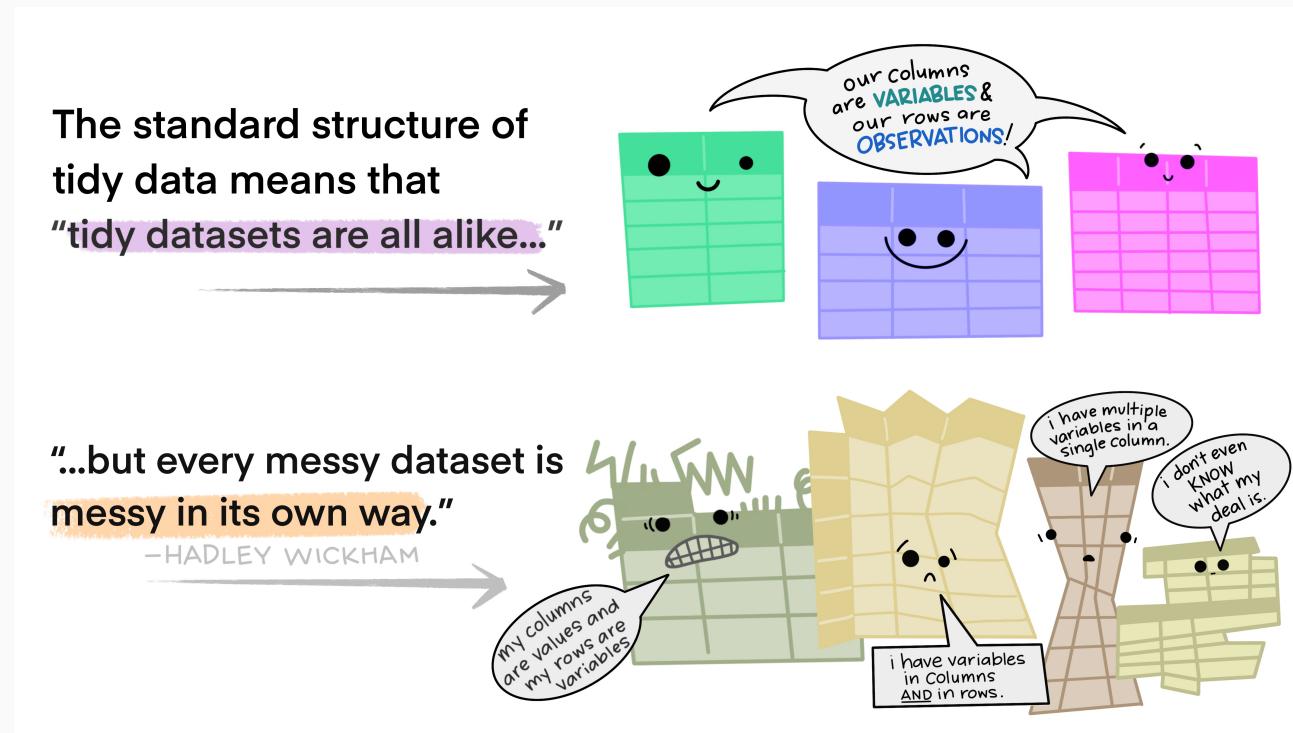
## Lösung

- Die Werte der Variable `bip/einwohner` werden aufgeteilt und den Variablen `bip` und `einwohner` zugewiesen.
- Bei Bedarf kann nun z. B. das BIP pro Kopf bestimmt werden (hier: `bip_pro_kopf`).

Beispiel **tidy**:

land	bip	einwohner	bip_pro_kopf
a	1000000	5000	200
b	1100000	1100000	176

# Grundlegendes Prinzip: ordentliche Daten (*tidy data*)



Quelle: Allison Horst

- Leseempfehlungen:
  - Data Organization in Spreadsheets (Broman/Woo ,2018)
  - Tidy Data (Wickham, 2014)

# Installation und Laden des Tidyverse

- Das Tidyverse besteht aus einer Sammlung von Paketen.
- Wir installieren es **ein Mal** mit dem Befehl:

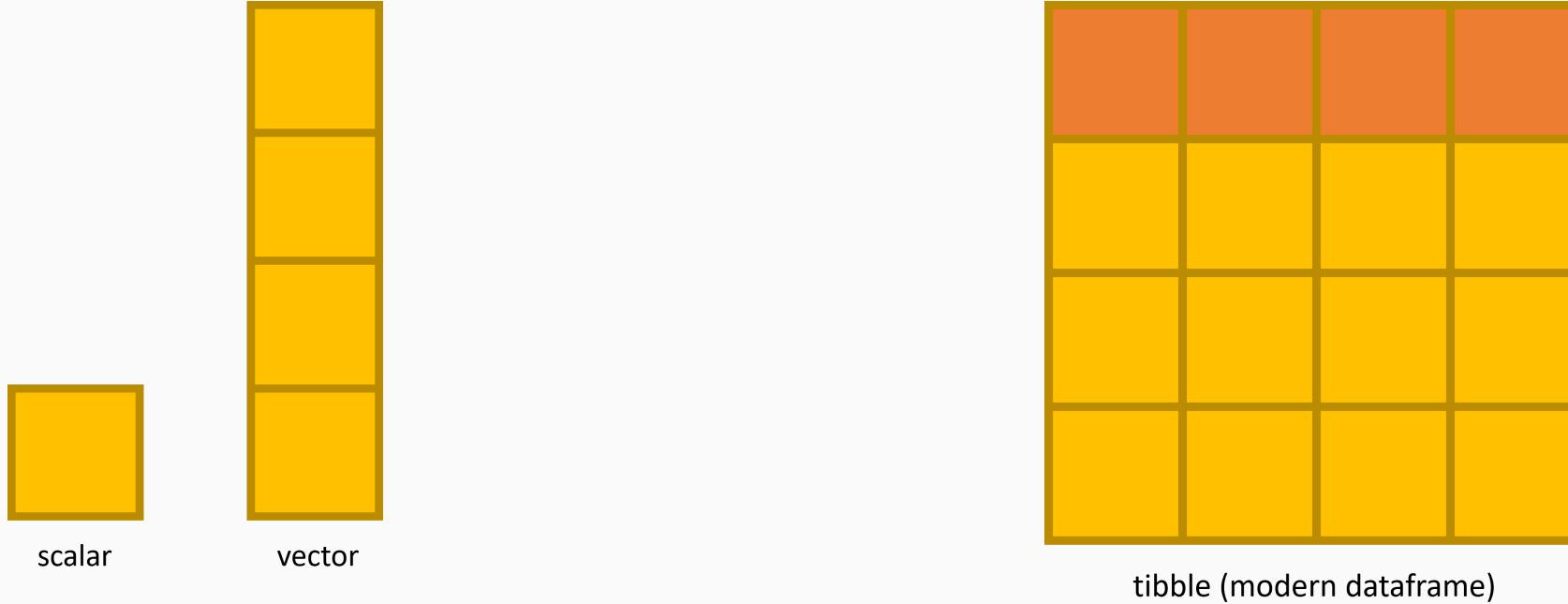
```
install.packages("tidyverse")
```

- Wir werden dieses Paket in Zukunft **zu Beginn jeder Datenanalyse** laden:

```
library(tidyverse)
```

# Die Datenstruktur im Tidyverse

- Eine Tabelle heißt im Tidyverse **tibble**.



- Ein Unterschied zu Skalaren und Vektoren ist die Kopfzeile, die den Variablennamen enthält.
- Es gibt auch **tibbles**, die nur eine Spalte und mehrere Zeilen oder sogar nur eine Spalte und eine Zeile enthalten.
- Die Regel sind jedoch Tabellen mit mehreren Spalten (Variablen) und mehreren Zeilen (Beobachtungen).

# Tibbles selbst erstellen

- Tibbles sind Kombinationen aus Vektoren, denen wir einen Namen zuweisen. z. B.:

```
land_bip_einwohner <- tibble(land = c("FantasiAland", "FantasiBland", "FantasiCland"), bip = c(1e+06, 11  
1020000), einwohner = c(5000, 6250, 4000), kontinent = c("EurAsien", "EurAsien", "EurBsien"))
```

```
land_bip_einwohner
```

```
#> # A tibble: 3 x 4  
#>   land           bip  einwohner kontinent  
#>   <chr>        <dbl>    <dbl> <chr>  
#> 1 FantasiAland 1000000      5000 EurAsien  
#> 2 FantasiBland 1100000     6250 EurAsien  
#> 3 FantasiCland 1020000     4000 EurBsien
```

- In der Konsole werden uns interessante Hinweise zum **tibble** angezeigt, z. B. die Dimension (hier: `3x3`) und die Variablentypen (hier: `character` und `double`).
- Im Environment werden **tibbles** als Objekttyp `tbl_df` angezeigt.

# Data Wrangling

(Daten bändigen)



HORST '19

Quelle: [Allison Horst](#)

# Fünf Verben für die Datenanalyse

- Die grundlegenden Schritte der Datenanalyse können mit **fünf Verben** durchgeführt werden.



Quelle: [rstudio/hex-stickers](#)

`mutate()` : fügt neue Variablen hinzu, diese können auf vorhandenen Variablen basieren.

`select()` : wählt Variablen auf Basis der Namen aus.

`filter()` : filtert Beobachtungen auf Basis ihrer Werte.

`summarise()` : fasst mehrere Werte zu einem einzelnen komprimierten Wert zusammen.

`arrange()` : verändert die Reihenfolge der Zeilen.

- Hinzu kommt:

`group_by()` : in Kombination mit einem der obigen Verben, können Operationen **je Gruppe** ausgeführt werden.

# Fünf Verben für die Datenanalyse

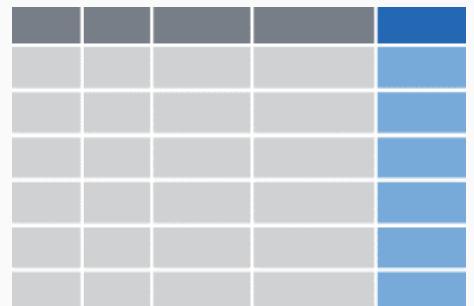
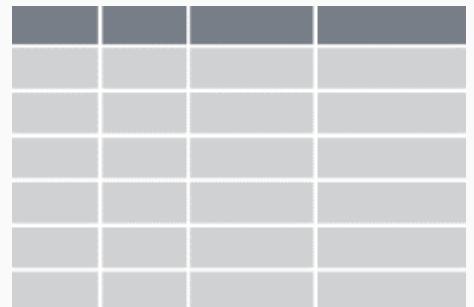
## mutate()

- Das erste Element, das den fünf Verben (Funktionen) übergeben werden muss, ist ein **tibble**.
- `mutate` wird als zweites Element ein Variablenname und eine Funktion übergeben, z. B.:

```
mutate(land_bip_einwohner, bip_pro_kopf = bip/einwohner)
```

```
#> # A tibble: 3 x 5
#>   land          bip  einwohner kontinent bip_pro_kopf
#>   <chr>        <dbl>    <dbl> <chr>        <dbl>
#> 1 FantasiAland 1000000     5000 EurAsien      200
#> 2 FantasiBland 1100000     6250 EurAsien      176
#> 3 FantasiCland 1020000     4000 EurBsien     255
```

Stilisierte  
Funktionsweise:



# Fünf Verben für die Datenanalyse

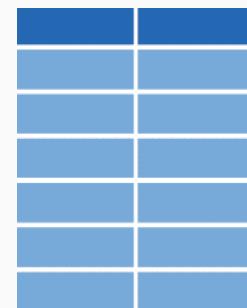
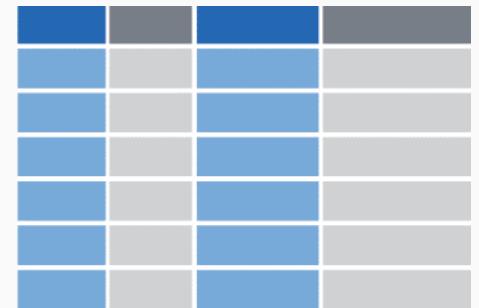
## select()

- `select` wird als zweites Element ein Vektor mit Variablennamen übergeben, z. B.:

```
select(land_bip_einwohner, c(land, einwohner))
```

```
#> # A tibble: 3 x 2
#>   land      einwohner
#>   <chr>     <dbl>
#> 1 FantasiAland    5000
#> 2 FantasiBland    6250
#> 3 FantasiCland    4000
```

Stilisierte  
Funktionsweise:



# Fünf Verben für die Datenanalyse

## filter()

- Im Falle von `filter()` folgt als zweites Argument ein logischer Test, z. B.:

```
filter(land_bip_einwohner, bip > 1000000)
```

```
#> # A tibble: 2 x 4
#>   land           bip  einwohner kontinent
#>   <chr>        <dbl>    <dbl> <chr>
#> 1 FantasiBland 1100000     6250 EurAsien
#> 2 FantasiCland 1020000     4000 EurBsien
```

Stilisierte  
Funktionsweise:





# Fünf Verben für die Datenanalyse

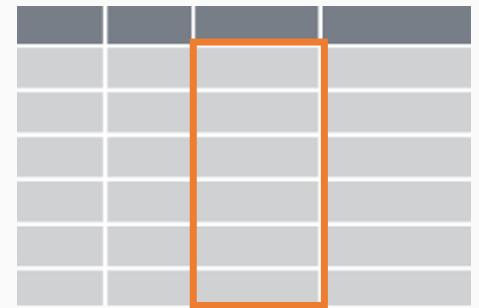
## summarise()

- Im Falle von `summarise()` folgt als zweites Argument eine zusammenfassende Funktion, z. B.:

```
summarise(land_bip_einwohner, durchschnitt_bip = mean(bip))
```

```
#> # A tibble: 1 x 1
#>   durchschnitt_bip
#>   <dbl>
#> 1 1040000
```

Stilisierte  
Funktionsweise:



# Exkurs: Funktionen

Everything that happens in R is a function. ~John M. Chambers

- Funktionen sind kleine Programme, die meistens aus einem **Input** einen **Output** generieren.
- **Input** ist alles, was **in** eine Funktion einfließt:
  - **Arguments** sind vordefinierte **Input**-Typen.
  - Den **Arguments** weisen wir **Values** zu.
- Stilisierte Funktionsweise:

Funktionsname(argument = value)

- Beispiel:

```
mean(x = c(100,200,NA), na.rm = TRUE))
```

# Funktionen

- R-Funktionen werden wie in der Mathematik von innen nach außen gelöst, z. B.:

$$f(g(x)),$$

mit  $f(x) = x^2$  und  $g(x) = x + 1$ . Dann gilt:

$$f(g(x)) = f(x + 1) = (x + 1)^2$$

.

- Dies wird insbesondere bei der Datenanalyse mit den fünf Verben relevant.
- Denn bei der explorativen Datenanalyse werden häufig verschiedene Datenanalyseschritte hintereinander durchgeführt.

## Funktionen aneinander reihen (klassisch)

- Wollen wir in unserem Beispieldatensatz zuerst nur Beobachtungen mit einem BIP von mehr als 1.000.000 filtern, dann eine neue Variable `bip_pro_kopf` erstellen und dann den Durchschnitt bestimmen, so könnte man Folgendes schreiben:

```
summarise(mutate(filter(land_bip_einwohner, bip > 1000000), bip_pro_kopf = bip/einwohner),  
durschn_bip_pro_kopf = mean(bip_pro_kopf))
```

- Dieser Code ist nicht nur schwer lesbar, sondern lässt sich auch schwer schreiben.



- Das Tidyverse bietet eine besser les- und schreibbare Alternative zu Coden, mit der **Pipe**:

## Funktionen aneinander reihen mit %>% (modern)

```
land_bip_einwohner %>%
  filter(bip > 1000000) %>%
  mutate(bip_pro_kopf = bip/einwohner) %>%
  summarise(durschn_bip_pro_kopf = mean(bip_pro_kopf))
```

- Die Pipe (%>%) liest sich als **»und dann«**.
- Der Code wird, wie Text sonst auch, von oben nach unten und von links nach rechts gelesen.
- Wir starten mit dem tibble `land_bip_einwohner` **und dann** filtern wir **und dann** erstellen wir eine neue Variable **und dann** berechnen wir den Durchschnitt.

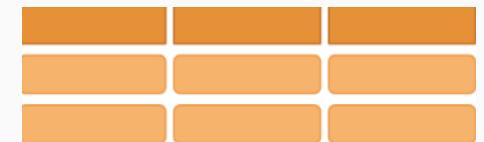
## Weiteres Beispiel: `group_by()` in Kombination mit `summarise()`

- Mit `group_by()` werden Operationen je Gruppe durchgeführt. In Kombination mit `summarise()` lassen sich z. B. Durchschnitte je Gruppe bestimmen:

```
land_bip_einwohner %>%  
  group_by(kontinent) %>%  
  summarise(durchschnitt_bip = mean(bip), durchschnitt_einwohner = mean(einv
```

```
#> # A tibble: 2 x 3  
#>   kontinent  durchschnitt_bip  durchschnitt_einwohner  
#>   <chr>          <dbl>                <dbl>  
#> 1 EurAsien      1050000              5625  
#> 2 EurBsien      1020000              4000
```

Stilisierte  
Funktionsweise:



Genug der Theorie. Ab nach

