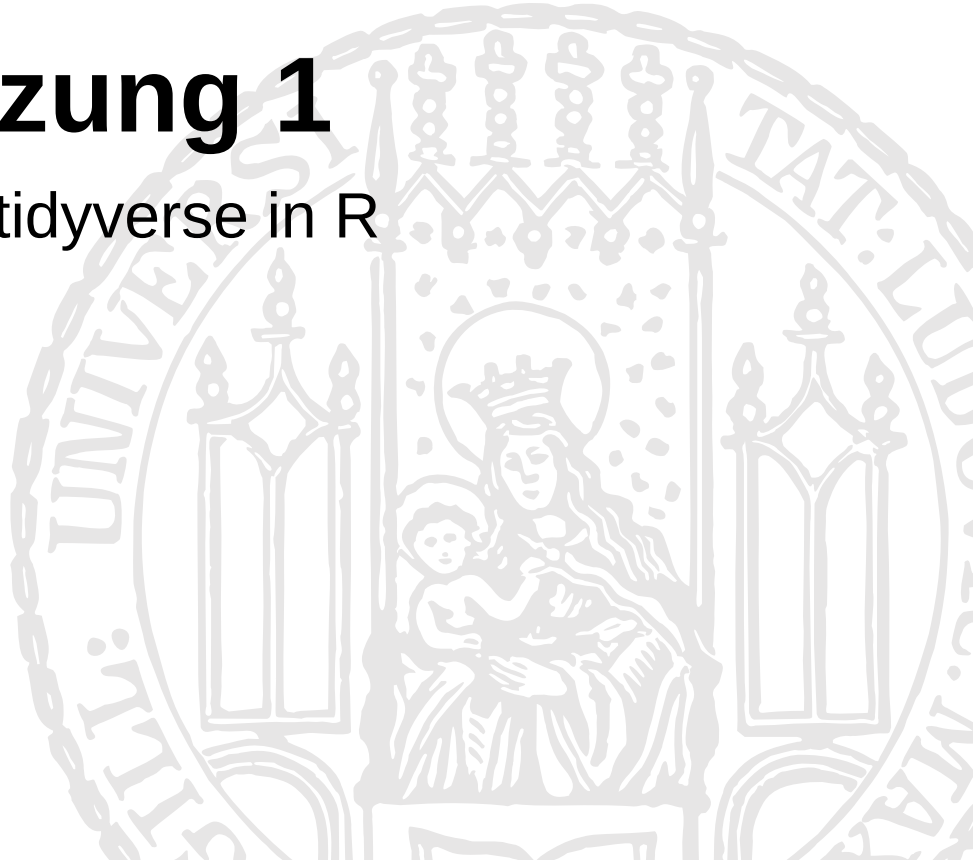
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Datenanalyse – Sitzung 1

Einführung in das Datenmanagement mit tidyverse in R

Institut für Kommunikationswissenschaft und Medienforschung
Ludwig-Maximilians-Universität München



Ablauf der Sitzung

1. Was sind Packages in R und wie installiere ich sie?
2. Übung 1: Packages von CRAN installieren und aktivieren
3. Warum Tidyverse statt Base R?
4. Packages im Tidyverse
5. Der Pipe-Operator %>%
6. Das kann das dplyr-Package: Funktionen für die Datenmodifikation

1. Was sind Packages in R und wie installiere ich sie?

Was sind Packages in R?

Packages sind Sammlungen von themenspezifischen Funktionen.

- Die vorinstallierte Version von R *(=* **`Base R`***)* besitzt vorinstallierte Grundfunktionen

  → einige Grundfunktionen kennen Sie bereits, z.B.: **`print()`**, **`c()`** und **`cbind()`**

- Packages erweitern die in Base R vorhandenen Grundfunktionen, d.h. sie machen R individualisierbar

  → eine solche Funktion haben Sie auch kennengelernt: **`as_tibble()`** stammt vom **`tidyverse`**-Package – deswegen haben wir diese Funktion letzte Woche auch noch nicht aktiv genutzt, sondern nur über sie gesprochen

Was sind Packages in R?

Wo findet man R-Packages?

- Packages sind auf CRAN (Comprehensive R Archive Network) verfügbar
- CRAN ist das Hauptrepository* für R-Packages

  → z.B. werden hier auch die Installationsdateien für `tidyverse` zur Verfügung gestellt – ein Package, das wir in dieser Sitzung benutzen werden

*Speicherort (oft online), wo Daten, Code oder Software für Dritte zum Download bereitgestellt werden

Woher kommen die R-Packages auf CRAN?

Forscher*innen und Entwickler*innen können zur Verbesserung von R beitragen, indem sie eigene R-Packages erstellen und auf CRAN für andere zugänglich machen. So ist auch `tidycomm` entstanden – unser hauseigenes Package für die Studierenden des IfKWs!

Wie kann ich den Umgang mit neuen Packages erlernen?

- Am leichtesten geht das über die Referenzhandbücher auf CRAN: Für jedes Package gibt es ein Referenzhandbuch, das dessen Funktionen auflistet und erklärt
- Meist enthalten sie sog. Vignetten – detaillierte Anleitungen mit konkreten Anwendungsbeispielen zur Verwendung der Funktionen eines Packages

Wie findet man diese Referenzhandbücher?

Der einfachste Weg ist die Verwendung einer Suchmaschine mit dem Suchbefehl, z.B. „CRAN tidycomm“, um direkt zum entsprechenden Package auf CRAN zu gelangen

Beispiel: CRAN-Seite für das „tidycomm“ Package

Abbildung 1. Eingabe in der Suchmaschine

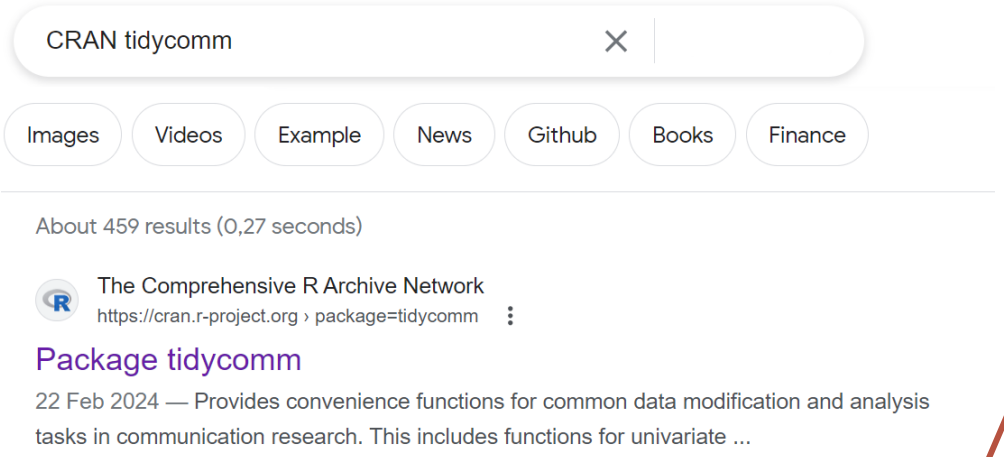
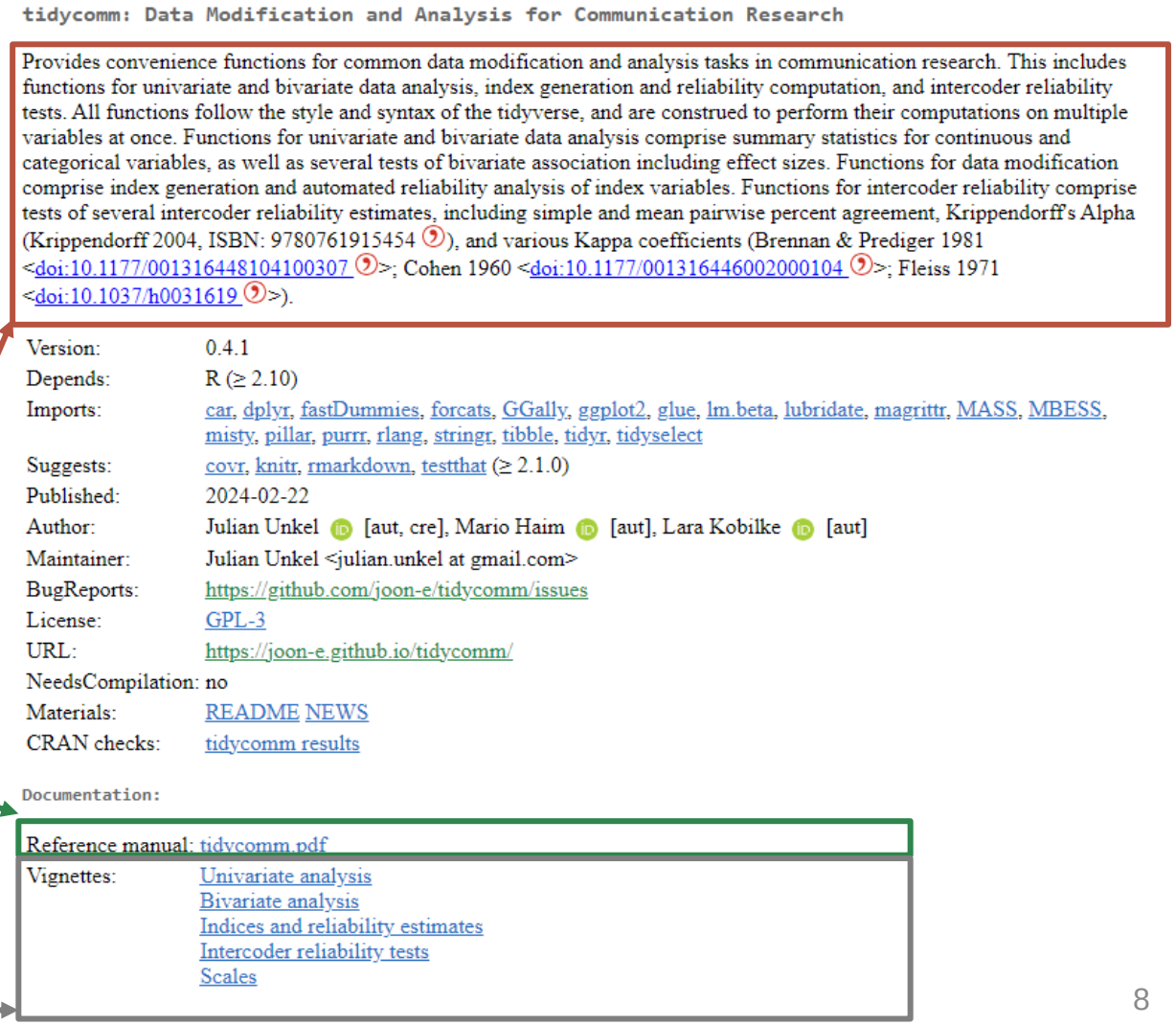


Abbildung 2. CRAN-Seite des „tidycomm“-Package



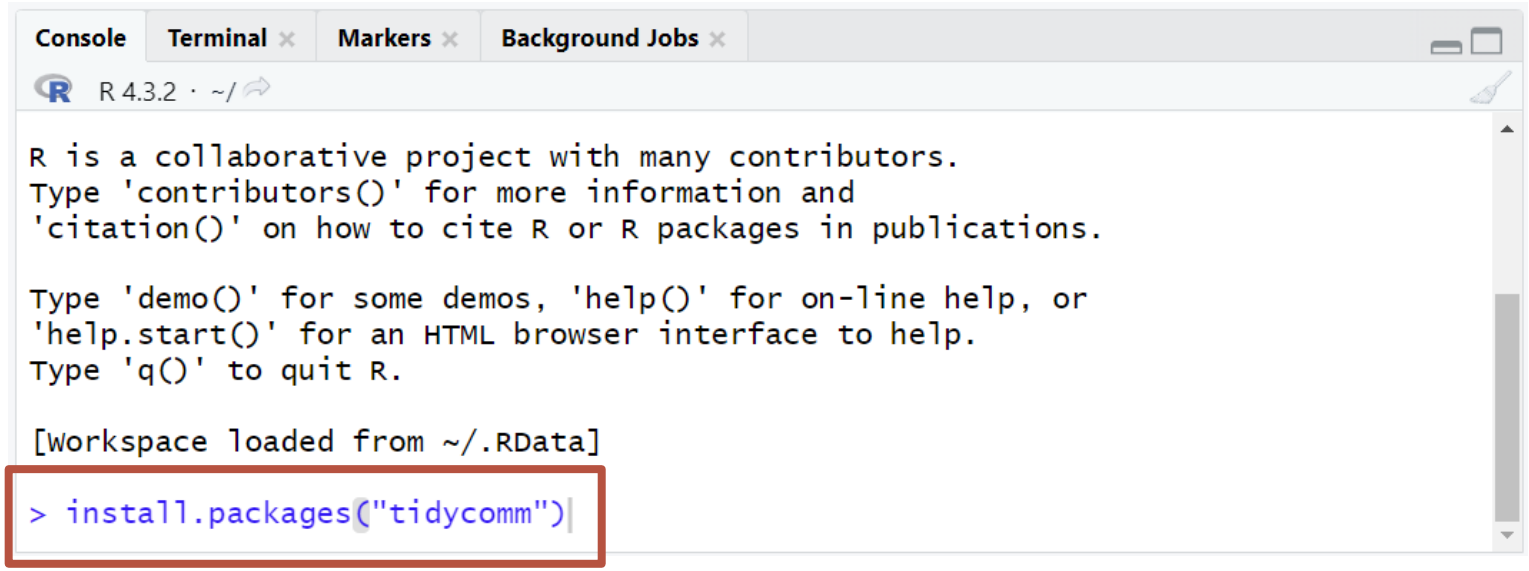
1. Allgemeine Informationen über Sinn, Zweck und Anwendungsfälle des Packages.

2. Link zum Referenzhandbuch, das die durch das Package zu Base R hinzugefügten Funktionen auflistet und erläutert.

3. Die Vignetten enthalten konkrete Beispiele, in denen die Funktionen an zugänglichen Datensätzen vorgeführt werden. Die schnellste Anlaufstelle, um ein neues Package zu erlernen!

Wie installiere ich ein Package in R?

Das geht ganz einfach in RStudio. Die Funktion dazu heißt **`install.packages()`**. Schreiben Sie die Funktion mitsamt Namen des Packages (i.d.R. in die Console) und drücken Sie dann Eingabe / Run, um den Befehl auszuführen.



Beachten Sie: Der Name des CRAN-Packages muss in Anführungszeichen geschrieben werden bei der ersten Installation via **`install.packages()`**!

Das Ergebnis der Package-Installation…

…sollte in etwa so aussehen (abhängig vom Betriebssystem Windows vs. MAC):

```
> install.packages("tidycomm")
Installiere Paket nach 'C:/Users/LaraK/AppData/Local/R/win-library/4.3'
(da 'lib' nicht spezifiziert)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.3/tidycomm_0.4.1.zip'
Content type 'application/zip' length 549942 bytes (537 KB)
downloaded 537 KB

Paket 'tidycomm' erfolgreich ausgepackt und MD5 Summen abgeglichen

Die heruntergeladenen Binärpakete sind in
        C:\Users\LaraK\AppData\Local\Temp\Rtmps902RW\downloaded_packages
```

Hurra! Unsere installierte R-Version hat gerade neue Funktionen gelernt und ist damit vielseitiger geworden! 🙌

Packages aktivieren: Der Standard

Auch wenn Sie ein Package bereits mittels `install.packages()` installiert haben, müssen Sie mit **jedem Neustart (!)** von R / RStudio dennoch die Packages erneut aktivieren, um dessen Funktionen nutzen zu können. Zur Aktivierung eines bereits installierten Packages benutzen Sie die Funktion `library()`, die Sie i.d.R. am Beginn Ihres R-Skripts platzieren.

Das neu installierte `tidycomm`-Package aktivieren Sie beispielsweise so:

```
library(tidycomm)
```

Beachten Sie: Keine Anführungszeichen bei der Aktivierung eines R-Packages in der aktuellen R-Sitzung mittels `library()`!

Packages aktivieren: Eine Alternative

Manchmal wollen Sie vielleicht nicht das ganze Package aktivieren, besonders wenn Sie nur eine Funktion daraus benötigen oder Konflikte mit Funktionen aus anderen Packages vermeiden möchten. In diesem Fall können Sie die Funktion direkt mit dem Paketnamen und dem Operator `::` aufrufen:

```
tidycomm::describe()
```

Das bedeutet:

- tidycomm: Das ist der Name des Packages.
- :: : Dieser Operator wird verwendet, um anzugeben, dass Sie eine Funktion aus einem spezifischen Package verwenden möchten.
- describe(): Das ist die Funktion, die Sie aus dem Package ausführen möchten.

→ Importiert also nur die Funktion `describe()` aus dem Package `tidycomm` für die aktuelle Operation, ohne das gesamte Package zu aktivieren

2. Übung 1

Packages von CRAN installieren und aktivieren

Übung 1: Packages installieren und aktivieren

Frage 1: Sie möchten das Package "ggplot2" installieren, um Grafiken in R herstellen zu können. Welcher Befehl wäre der richtige?

A) `install.packages(ggplot2)`

B) `install.packages("ggplot2")`

C) `library("ggplot2")`

D) `library(ggplot2)`

Lösung für Übung 1: Packages installieren und aktivieren

Frage 1: Sie möchten das Package "ggplot2" installieren, um Grafiken in R herstellen zu können. Welcher Befehl wäre der richtige?

B) `install.packages("ggplot2")`

Übung 1: Packages installieren und aktivieren

Frage 2: Was passiert, wenn Sie `library(ggplot2)` ausführen, nachdem Sie das Paket mit `install.packages("ggplot2")` installiert haben?

A) Das Paket wird erneut installiert.

B) Das Paket wird geladen und ist bereit zur Verwendung.

C) Es wird eine Fehlermeldung angezeigt, da der Befehl falsch ist.

D) Nichts passiert.

Lösung für Übung 1: Packages installieren und aktivieren

Frage 2: Was passiert, wenn Sie `library(ggplot2)` ausführen, nachdem Sie das Paket mit `install.packages("ggplot2")` installiert haben?

B) Das Paket wird geladen und ist bereit zur Verwendung.

Übung 1: Packages installieren und aktivieren

Frage 3: Was schätzen Sie, bedeutet die Fehlermeldung „Error in library(ggplot2) : there is no package called ‘ggplot2’“?

A) Sie haben das Paket bereits installiert.

B) Sie haben den Paketnamen falsch geschrieben.

C) Das Paket ist nicht installiert und muss zuerst installiert werden.

D) Sie müssen R neu starten, bevor Sie das Paket verwenden können.

Lösung für Übung 1: Packages installieren und aktivieren

Frage 3: Was schätzen Sie, bedeutet die Fehlermeldung „Error in library(ggplot2) : there is no package called ‘ggplot2’“?

C) Das Paket ist nicht installiert und muss zuerst installiert werden.

Wichtige Take-Aways

- Packages installieren (nur ein einziges Mal): **`install.packages("")`**
- Installiertes Package aktivieren: **`library()`**

3. Warum Tidyverse statt Base R?

Daten laden: Aufgabe 1a) aus Sitzung 1

Erzeugen Sie zunächst noch einmal den data_aufg1a Datensatz aus der letzten Sitzung, mit dem wir nun weiterarbeiten wollen:

```r
# Daten von Sitzung 1, Aufgabe 1a) laden:
gruppe <- c("Experimentalgruppe", "Kontrollgruppe", "Kontrollgruppe", "Experimentalgruppe",
            "Experimentalgruppe", "Kontrollgruppe", "Kontrollgruppe", "Experimentalgruppe",
            "Experimentalgruppe", "Kontrollgruppe", "Kontrollgruppe", "Experimentalgruppe")
alter <- c(55, 21, 34, 64, 19, 47, 87, 25, 18, 63, 29, 45)
geschlecht <- c("maennlich", "maennlich", "maennlich", "weiblich", "weiblich", "weiblich",
                "maennlich", "maennlich", "maennlich", "weiblich", "weiblich", "weiblich")
mng_scholz <- c(1, 3, 4, 3, 5, 2, 1, 3, 4, 3, 5, 2)
mng_soeder <- c(2, 1, 5, 4, 4, 2, 2, 1, 5, 4, 4, 2)

data <- cbind(gruppe, alter, geschlecht, mng_scholz, mng_soeder)
```

Daten laden: Aufgabe 1a) aus Sitzung 1

Für die nachfolgenden Datentransformationen müssen wir das tidyverse-Package laden und die data_aufg1a-Daten als Tibble formatieren

```
14  library(tidyverse)
15  data <- as_tibble(data)
16  print(data)
```

Ausgabe in der Console:

```
# A tibble: 12 × 5
   gruppe              alter geschlecht mng_scholz mng_soeder
   <chr>               <chr> <chr>      <chr>      <chr>
 1 Experimentalgruppe  55    maennlich  1          2
 2 Kontrollgruppe      21    maennlich  3          1
 3 Kontrollgruppe      34    maennlich  4          5
 4 Experimentalgruppe  64    weiblich   3          4
 5 Experimentalgruppe  19    weiblich   5          4
 6 Kontrollgruppe      47    weiblich   2          2
 7 Kontrollgruppe      87    maennlich  1          2
 8 Experimentalgruppe  25    maennlich  3          1
 9 Experimentalgruppe  18    maennlich  4          5
10 Kontrollgruppe      63    weiblich   3          4
11 Kontrollgruppe      29    weiblich   5          4
12 Experimentalgruppe  45    weiblich   2          2
```

Vergleich der Syntax

Base R

```
print(data)
```

```
# A tibble: 12 × 5
   gruppe              alter geschlecht mng_scholz
   <chr>               <chr> <chr>      <chr>
 1 Experimentalgruppe  55    maennlich  1
 2 Kontrollgruppe      21    maennlich  3
 3 Kontrollgruppe      34    maennlich  4
 4 Experimentalgruppe  64    weiblich   3
```

```
data[data$geschlecht == "maennlich", 2]
```

```
# A tibble: 6 × 1
  alter
  <chr>
1 55
2 21
3 34
4 87
```

Beide Codebeispiele machen exakt dasselbe:

1) Daten werden zunächst begutachtet mittels `print()`.
2) Die Variable „alter“ wird ausgewählt (Spalte 2).
3) Fälle werden angezeigt, die männlich sind (6).

Welche Unterschiede fallen Ihnen auf?

Tidyverse

```
library(tidyverse)

data %>%
print()
```

```
# A tibble: 12 × 5
   gruppe              alter geschlecht mng_scholz
   <chr>               <chr> <chr>      <chr>
 1 Experimentalgruppe  55    maennlich  1
 2 Kontrollgruppe      21    maennlich  3
 3 Kontrollgruppe      34    maennlich  4
 4 Experimentalgruppe  64    weiblich   3
```

```
data %>%
select(alter) %>%
filter(geschlecht == "maennlich")
```

```
# A tibble: 6 × 1
  alter
  <chr>
1 55
2 21
3 34
4 87
```

Tidyverse-Philosophie

- Ein kohärenter Satz von Funktionen mit intuitiver Syntax für Datenmanipulation und -analyse, der die Datenmanipulation, -visualisierung und -modellierung erleichtert.
- Schritt-für-Schritt-Vorgehen beim Schreiben des Codes, was die Lesbarkeit (von oben nach unten) und Wartbarkeit des Codes verbessert.
- Ziel ist es, die Datenanalyse zugänglicher, verständlicher und angenehmer zu gestalten.

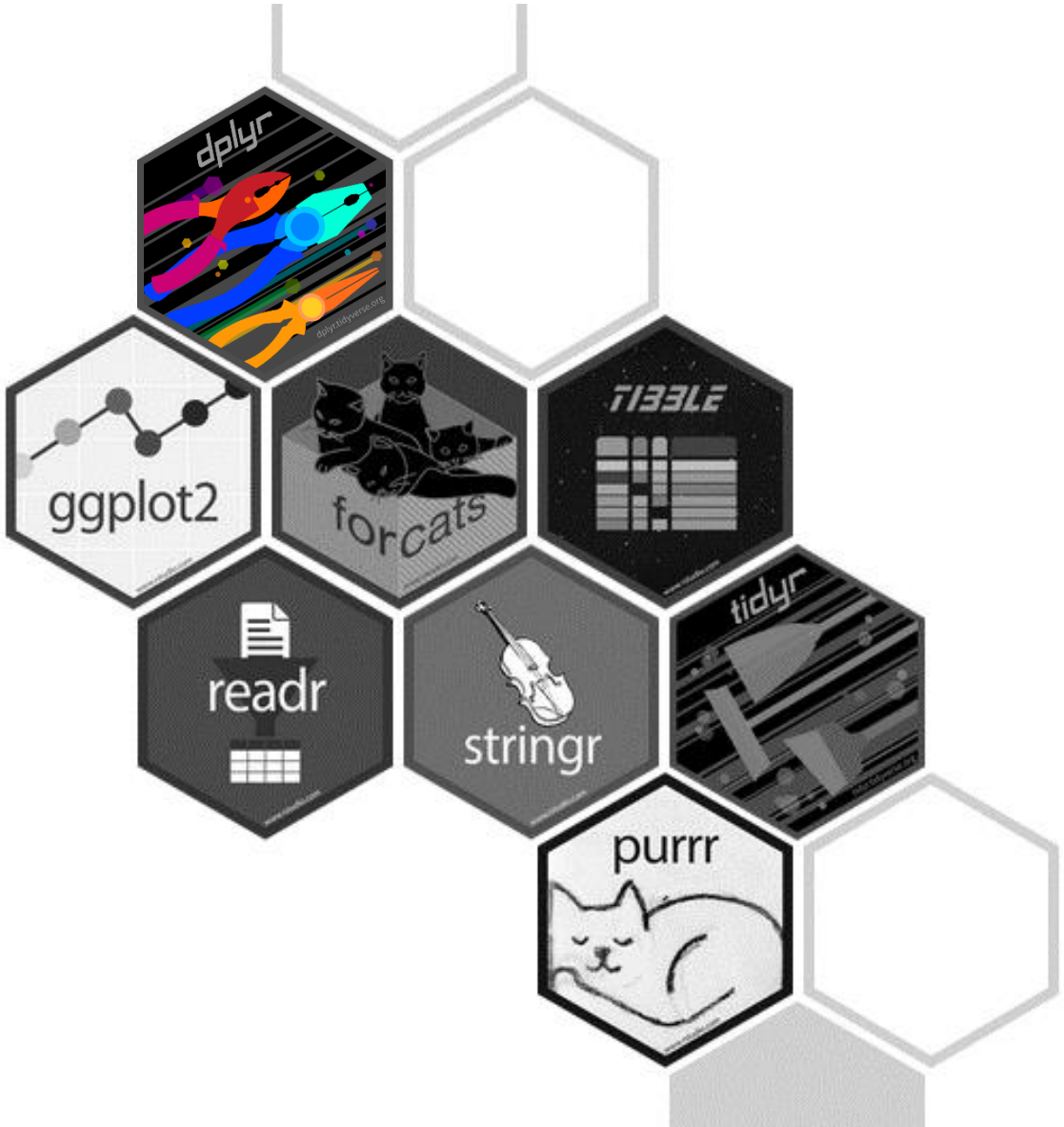
Das Tidyverse ist ein „Universum“ an Packages, die alle der gleichen „tidy“ Logik folgen. In seiner Summe ist das Tidyverse mächtig und verständlich, weshalb auch wir diesem Ansatz folgen.

4. Packages im Tidyverse

Packages im Tidyverse

Detaillierte Erklärungen, mehr Infos und… CHEATSHEETS ☺:
https://www.tidyverse.org/

Packages im Tidyverse

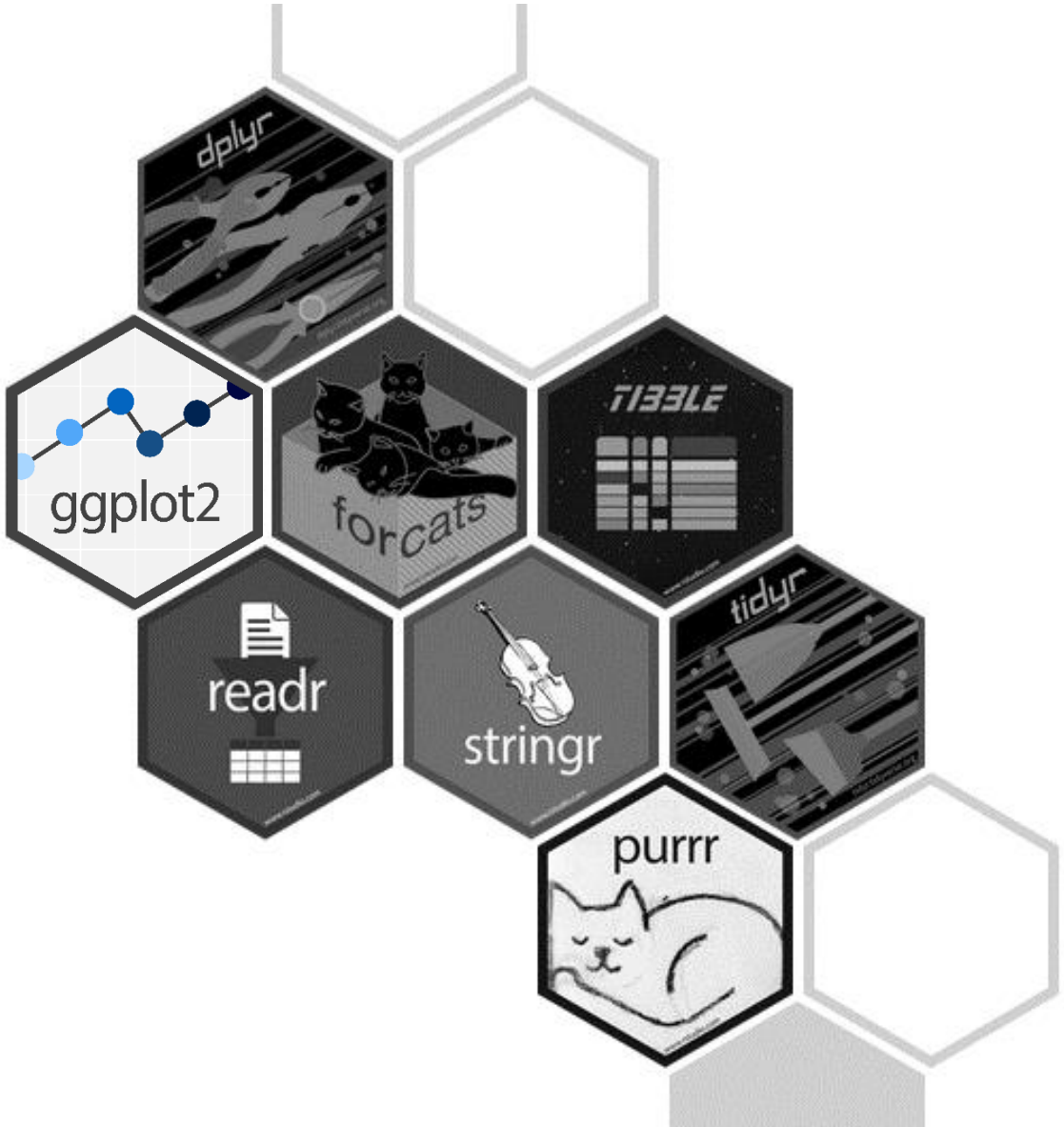


dplyr: Datenmanipulation

- Bietet eine Sammlung von Funktionen zur Bearbeitung von Datenrahmen: filtern, sortieren, zusammenfassen und mehr.
- Optimiert für Geschwindigkeit und Lesbarkeit.

→ Wichtig für uns. Einige zentrale und mächtige Funktionen daraus sehen wir uns gleich genauer an.

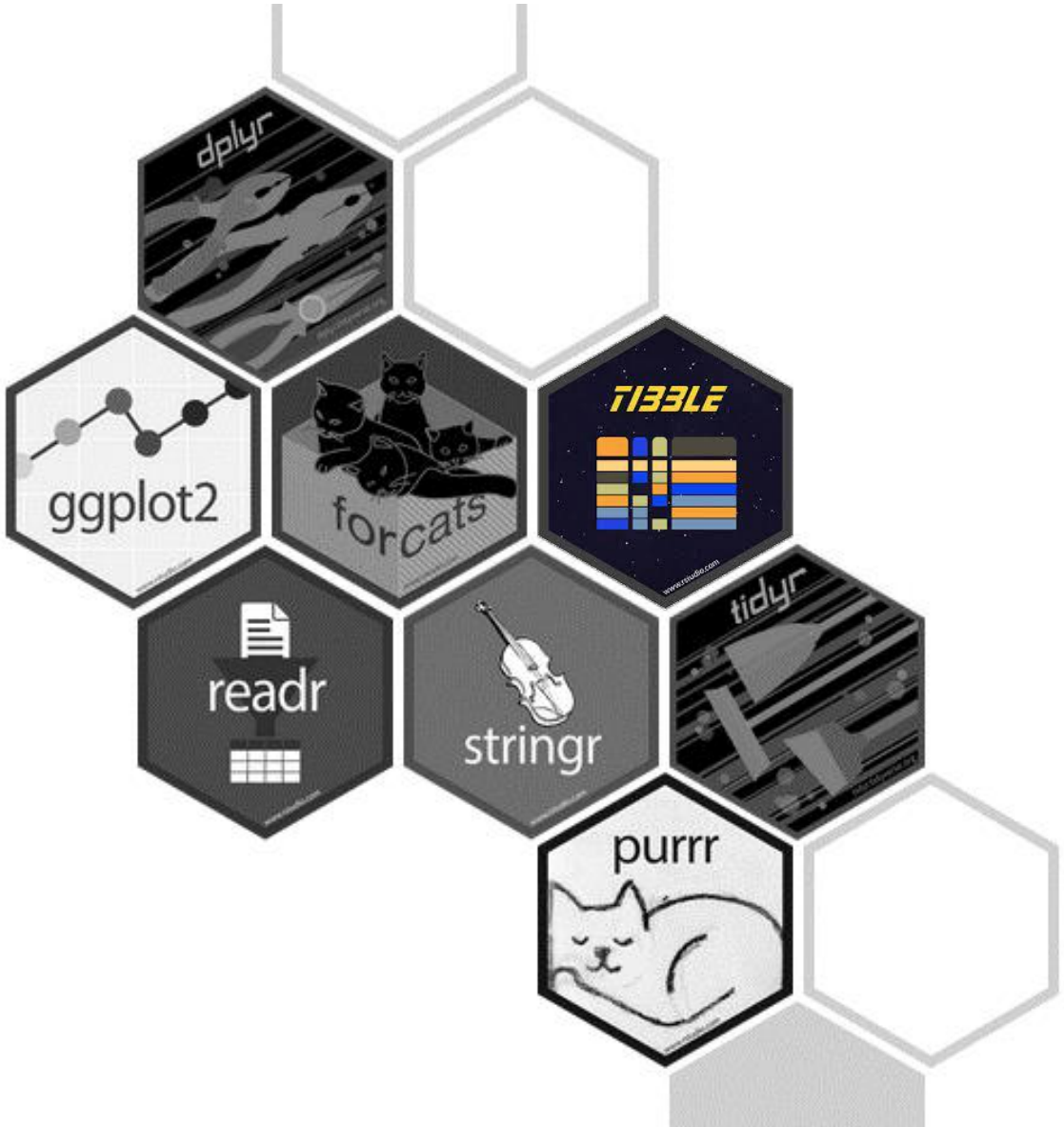
Packages im Tidyverse



ggplot2: Datenvisualisierung

- Erzeugt eine Vielzahl verschiedener, auch komplexer statistischer Grafiken.
- „Zerlegt“ Grafiken in unterschiedliche Bestandteile, um eine Vielzahl von Visualisierungsarten zu ermöglichen.

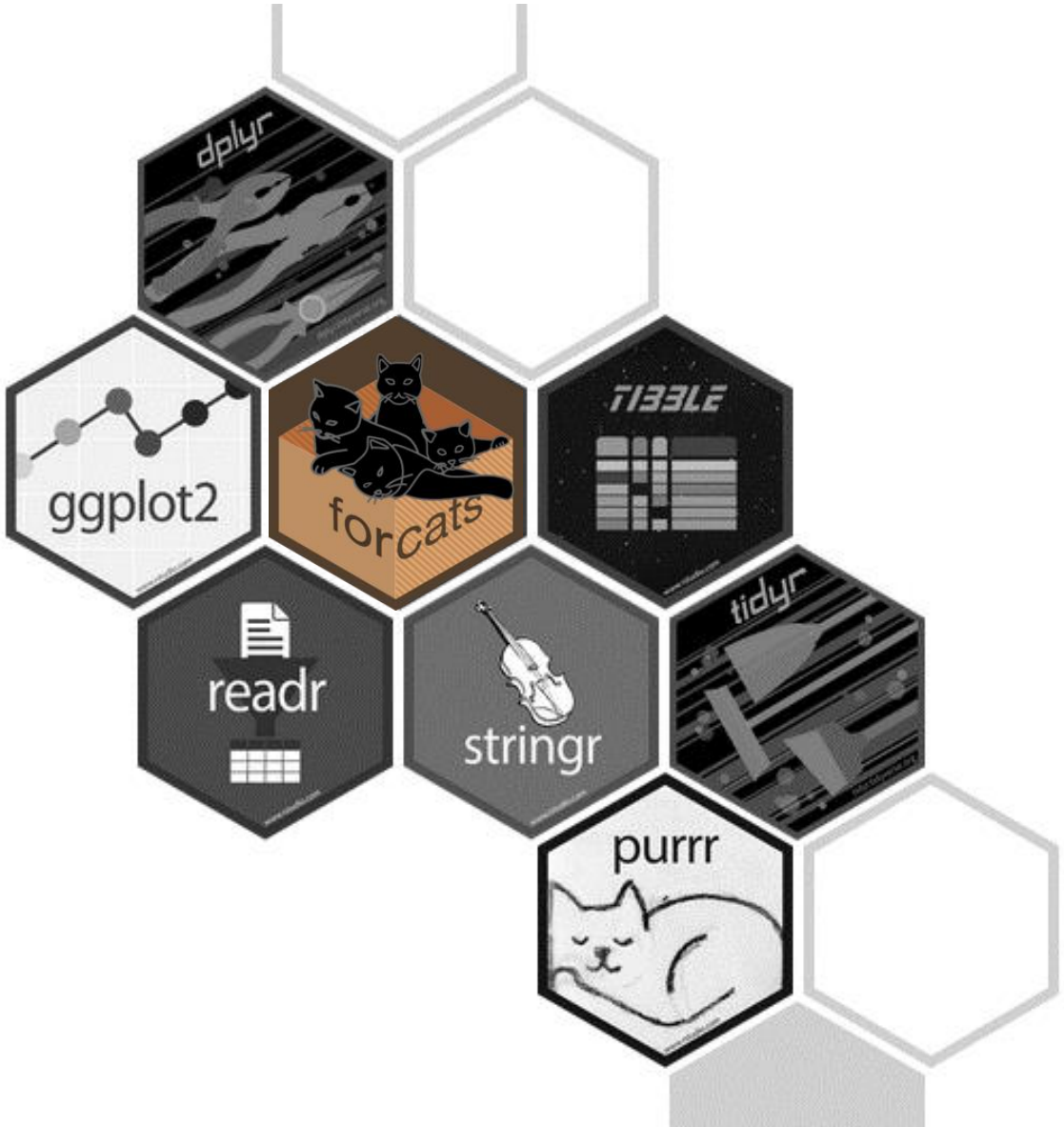
Packages im Tidyverse



tibble: Weiterentwicklung des Datenformats „Dataframe“ aus Base R

- Tibbles sind eine neuere, benutzerfreundlichere Form des Data Frames.
- Im Vergleich zum Dataframe ist der Hauptvorteil eines Tibbles, dass er übersichtlicher ausgegeben wird.

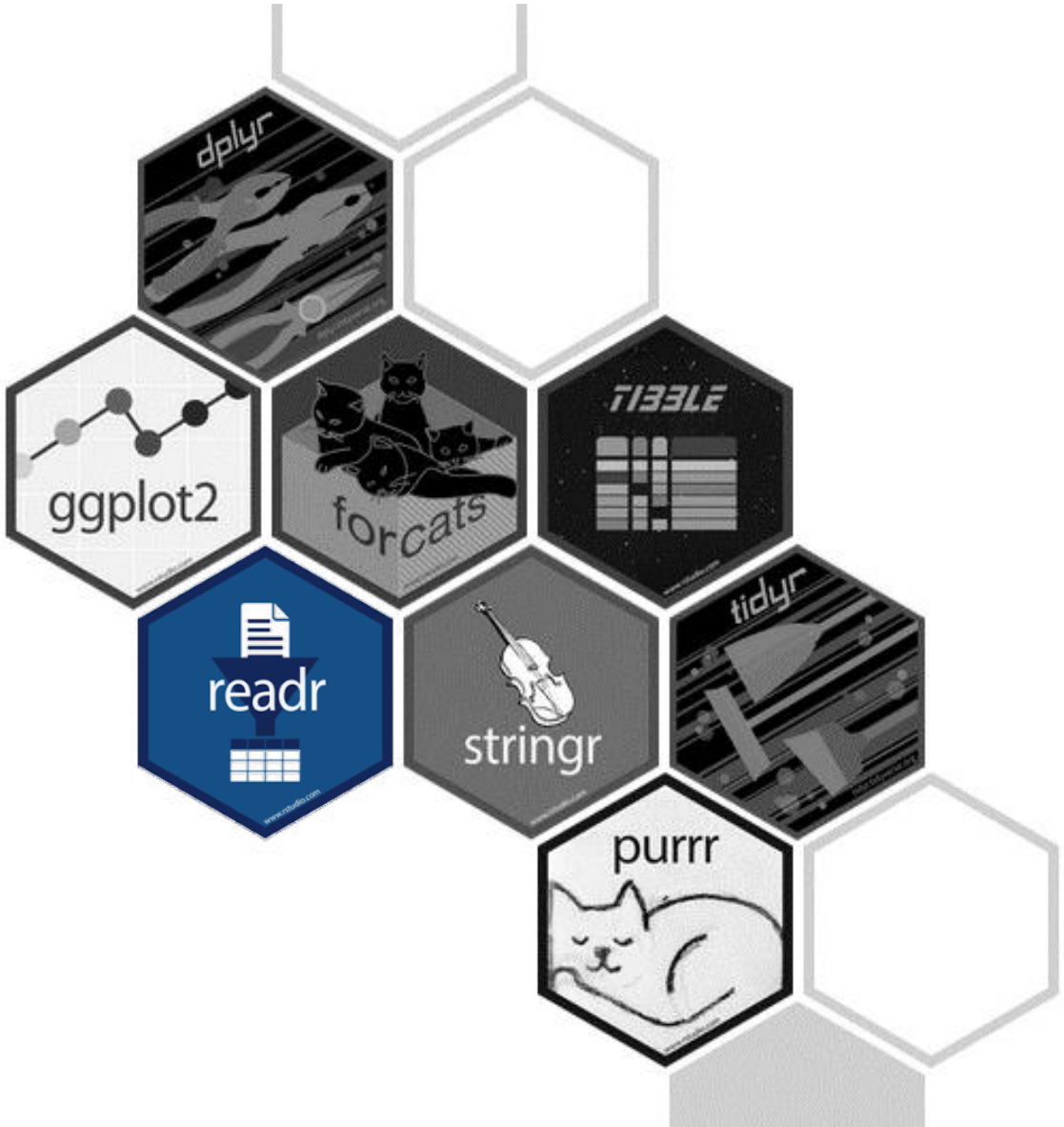
Packages im Tidyverse



forcats: Faktorenhandhabung

- Bietet Funktionen zur Arbeit mit nominalen und ordinalen Variablen (sog. „Faktoren“ in R).
- Ermöglicht einfache Umordnung, Umbenennung und Zusammenfassung von Faktorstufen.

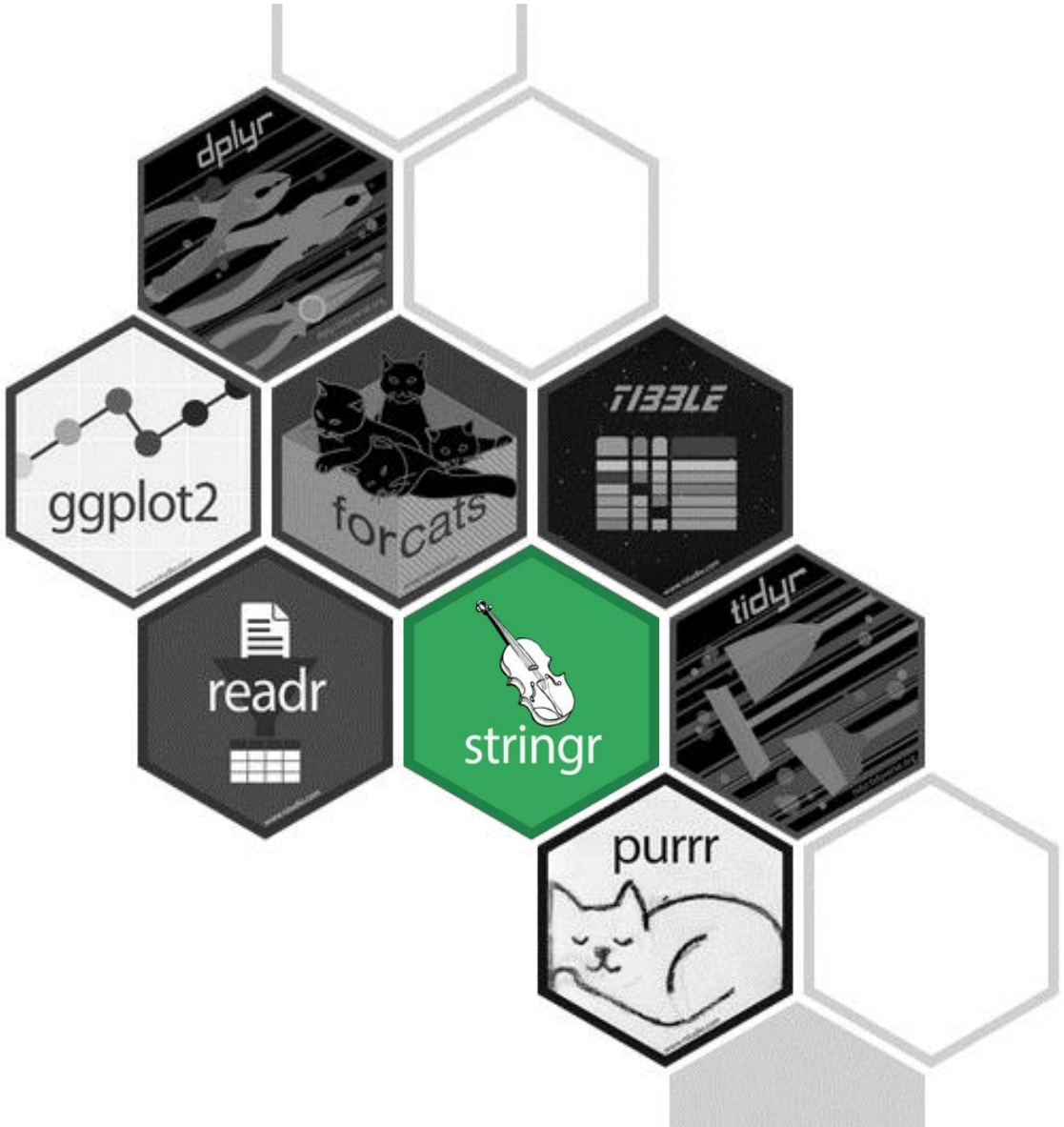
Packages im Tidyverse



readr: Datenimport

- Ermöglicht schnellen und einfachen Import von tabellarischen Daten aus Textdateien (CSV, TSV).

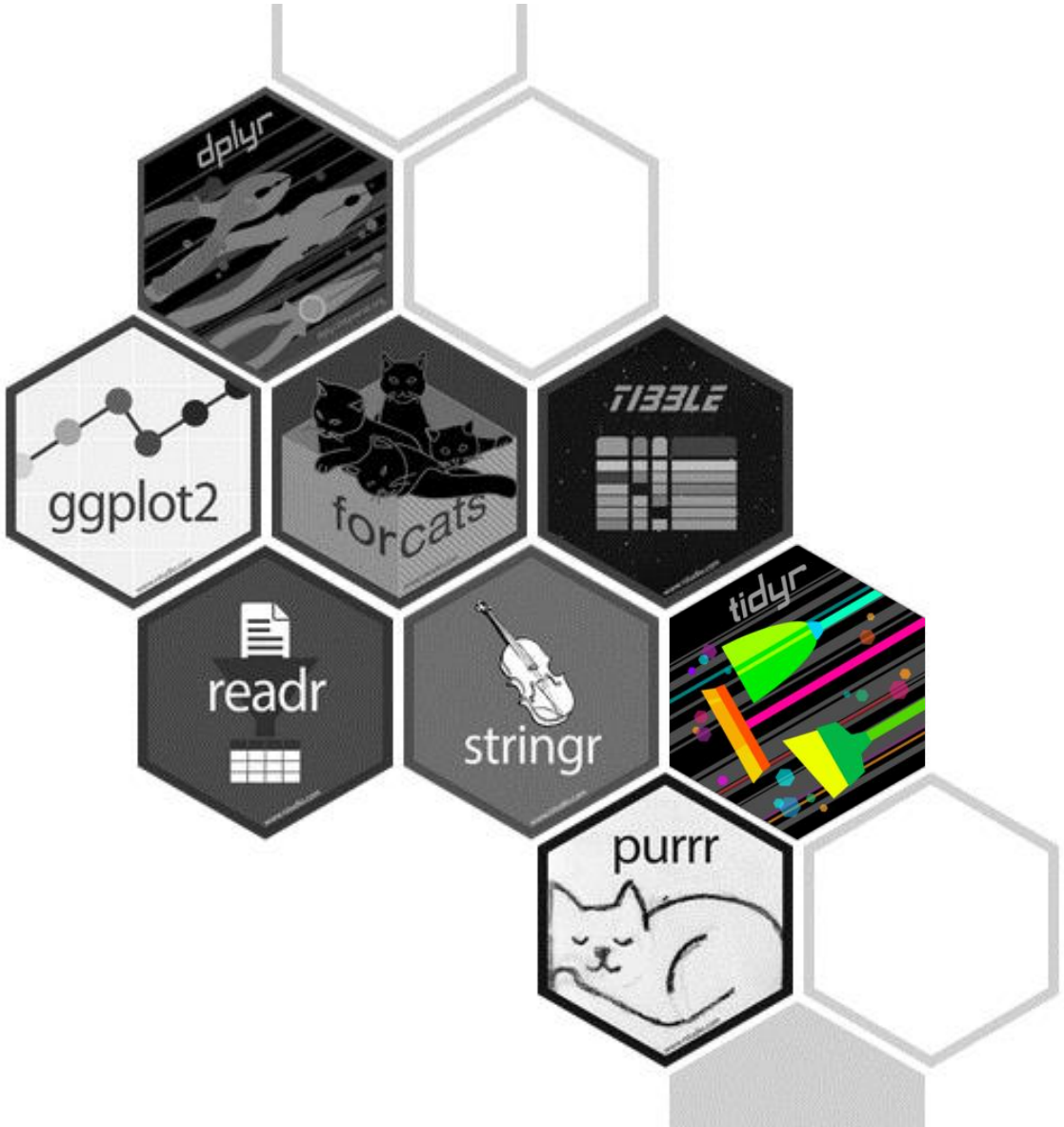
Packages im Tidyverse



stringr: Manipulation von Zeichenketten („Strings“)

- Vereinfacht die Arbeit mit Zeichenketten / Character-Variablen („Strings“), einschließlich der Manipulation und Bereinigung von Textdaten.

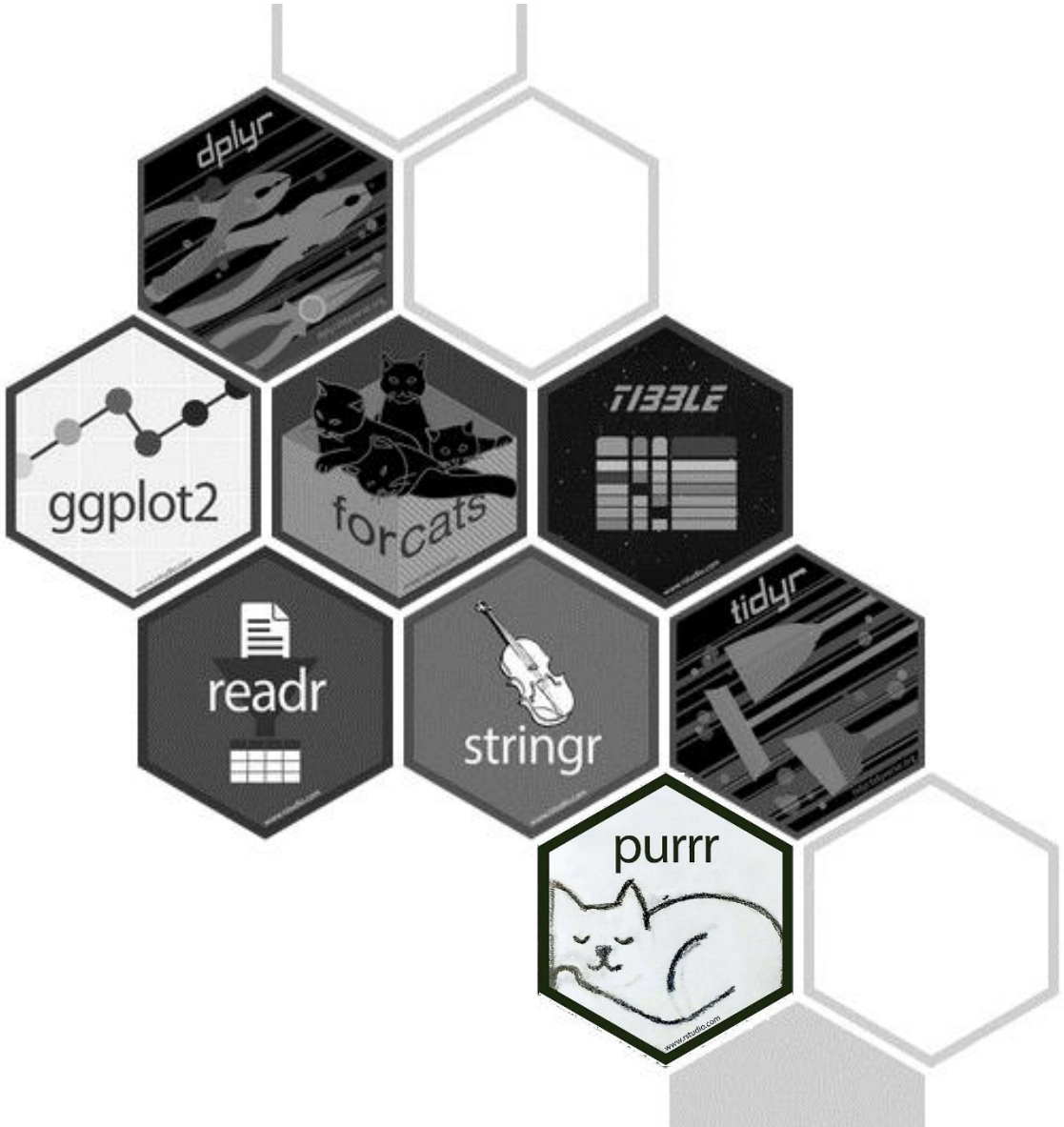
Packages im Tidyverse



tidyr: Datentransformation

- Hilft dabei, Daten zu „tidy data“, dem bevorzugten Datenformat im Tidyverse, das die Datenanalyse erleichtert, zu transformieren.
- Funktionen zum Umformen, Trennen und Verbinden von Datenspalten.

Packages im Tidyverse



purrr: Funktionale Programmierung

- Bietet Werkzeuge, um Funktionen effizient auf Listen und Vektoren anzuwenden.
- Ermöglicht es, komplexe Operationen mit weniger Code und mehr Klarheit auszuführen.

5. Der Pipe-Operator %>%

Der Pipe-Operator

%>%

Entweder tippen oder Shortcut verwenden.
Shortcut in R: Ctrl (Windows) / Cmd (Mac) + Shift + M

Vereinfacht gelesen als: „Und mach damit das Folgende…“

Vorteile:

- **Lesbarkeit**: durch weniger Verschachtelung und klare Schrittfolge in der Datenverarbeitung verbessert.
- **Wartbarkeit**: einfache Anpassung einzelner Schritte ohne große Strukturänderungen.
- **Effizienz**: in Entwicklung und Analyse.

Nimm mein R-Objekt „data“ **und mach damit das Folgende**:

Beispiel:

```
data %>%
  select(alter) %>%
  filter(geschlecht == "maennlich")
```

Wähle die Spalte „alter“ aus...

Und filtere (d.h. behalte) alle Fälle, bei denen die Befragten in der Spalte „Geschlecht“ „maennlich“ angegeben haben.

Ergebnis:

```
# A tibble:
  alter
  <chr>
1 55
2 21
3 34
4 87
```

Wichtige Take-Aways

- Tidyverse: Meta-Package mit intuitiver R-Syntax + Schritt-für-Schritt-Vorgehen beim Schreiben des Codes, was die Lesbarkeit (von oben nach unten) und Wartbarkeit des Codes verbessert
- Pipe-Operator („Und mach damit das Folgende…“): Verkettet Funktionen im Tidyverse. Shortcut: Ctrl (Windows) / Cmd (Mac) + Shift + M

6. Das kann das DPLYR-Package:

Funktionen für die Datenmodifikation

Wichtige Tidyverse-Funktionen zur Datenmodifikation

Im Tidyverse, speziell im Package **`dplyr`**, sind Funktionen enthalten, die (in der Kombination miteinander) sehr mächtig bei der Datenmodifikation sind:

- **`select()`**: Damit können aus einem bestehenden Datensatz bestimmte Variablen (Spalten) ausgewählt werden, um so die Breite des Datensatzes auf die interessierenden Variablen zu reduzieren.
- **`filter()`**: Damit können aus einem bestehenden Datensatz bestimmte Fälle (Zeilen) basierend auf einer Filterbedingung herausgefiltert werden.
- **`mutate()`**: Damit können Variablen hinzugefügt oder verändert werden, beispielsweise basierend auf einer mathematischen Umformungsbedingung.
- **`arrange()`**: Damit kann ein Datensatz auf Grundlage einer ausgewählten Variable nach der Größe auf- oder absteigend sortiert werden. *[aus Zeitgründen nicht bei uns im Datenanalyse-Seminar besprochen.]*

Bestimmte Variablen auswählen: `select()`

Bestimme Variablen auswählen mit `select()`

Struktur:

```
data %>%
     select(variable1, variable2, …)
```

Beispiel: Aus dem Datensatz data_aufg1a der letzten Sitzung werden zwei Variablen ausgewählt, die die Soziodemografie beschreiben (alter, geschlecht).

```
18  # select()
19  data %>%
20    select(alter, geschlecht)
```

`select()` (Fortsetzung Beispiel)

Ausgabe in der Console:

```
# A tibble: 12 × 2
   alter geschlecht
   <chr> <chr>
 1 55    maennlich
 2 21    maennlich
 3 34    maennlich
 4 64    weiblich
 5 19    weiblich
 6 47    weiblich
 7 87    maennlich
 8 25    maennlich
 9 18    maennlich
10 63    weiblich
11 29    weiblich
12 45    weiblich
```

`select()`: Erweiterte Selektion mithilfe von Helper Functions

Im Tidyverse sind einige **Helper Functions** implementiert, die die Auswahl spezifischer – beispielsweise ähnlicher – Variablen erleichtern.

Die wichtigsten davon sind:

- `starts_with("String")`
  → Auswahl von Spalten, die mit einem bestimmten String (= Zeichenkette) beginnen

- `ends_with("String")`
  → Auswahl von Spalten, die mit einem bestimmten String (= Zeichenkette) enden

- `contains("String")`
  → Auswahl von Spalten, die einen bestimmten String (= Zeichenkette) enthalten

`select()` mit Helper Function – Beispiel

In unserem Datensatz data_aufg1a sollen alle Variablen ausgewählt werden, in denen es um die Meinung zu Politikern geht. Man erkennt sie daran, dass diese mit „mng“ starten.

```
22  # select() mit Helper
23  data %>%
24    select(starts_with("mng"))
```

Ausgabe in der Console:

```
> data %>%
+   select(starts_with("mng"))
# A tibble: 12 × 2
   mng_scholz mng_soeder
   <chr>      <chr>
 1 1          2
 2 3          1
 3 4          5
 4 3          4
 5 5          4
 6 2          2
 7 1          2
 8 3          1
 9 4          5
10 3          4
11 5          4
12 2          2
```

Bestimmte Fälle aus dem Datensatz Herausfiltern:

`filter()`

Bestimme Fälle aus dem Datensatz herausfiltern mit `filter()`

Struktur:

```
data %>%
      filter(Filtervariable *Filterbedingung*)
```

Um die Filterbedingung zu setzen, benötigt man **Operatoren.** Die gängigsten sind:

- **>**: Größer als
- **<**: Kleiner als
- **==**: Gleich
- **!=**: Ungleich
- **>=**: Größer gleich
- **<=**: Kleiner gleich

- **%in%**: Enthalten in
  → gefolgt von einem Vektor oder einer Liste mit möglichen Werten
- **&**: Und
  → verknüpft zwei Bedingungen. Beide müssen zutreffen.
- **|**: Oder
  → verknüpft zwei Bedingungen. Mindestens eine davon muss zutreffen.

`filter()` (Beispiel 1)

Aus dem Datensatz data_aufg1a sollen nur Zeilen mit Daten über Befragten, die sich in der Experimentalgruppe befinden, herausgefiltert werden.

```
27  data %>%
28    filter(gruppe == "Experimentalgruppe")
```

Ausgabe in der Console:

```
> data %>%
+   filter(gruppe == "Experimentalgruppe")
# A tibble: 6 × 5
  gruppe              alter geschlecht mng_scholz mng_soeder
  <chr>               <chr> <chr>      <chr>      <chr>
1 Experimentalgruppe 55    maennlich  1          2
2 Experimentalgruppe 64    weiblich   3          4
3 Experimentalgruppe 19    weiblich   5          4
4 Experimentalgruppe 25    maennlich  3          1
5 Experimentalgruppe 18    maennlich  4          5
6 Experimentalgruppe 45    weiblich   2          2
```

Frage: Wie viele Befragte befinden sich in der Experimentalgruppe? Und woran können wir das erkennen? (Also: Wie viele Fälle sind im gefilterten Datensatz?)

`filter()` (Beispiel 2)

Aus data_aufg1a sollen nur Zeilen mit Daten über Befragte angezeigt werden, deren Meinung zu Scholz „sehr schlecht“ (1), „teils teils“ (3) oder „ sehr gut“ (5) ist.

Möglichkeit 1:

```
31  data %>%
32    filter(mng_scholz == 1 | mng_scholz == 3 | mng_scholz == 5)
```

Möglichkeit 2:

```
34  data %>%
35    filter(mng_scholz %in% c(1, 3, 5))
```

Möglichkeit 3:

```
37  data %>%
38    filter(mng_scholz != 2 & mng_scholz != 4)
```

`filter()` (Beispiel 2 - Fortsetzung)

Ausgabe in der Console (in allen Fällen):

```
# A tibble: 8 × 5
  gruppe              alter geschlecht mng_scholz mng_soeder
  <chr>               <chr> <chr>      <chr>      <chr>
1 Experimentalgruppe  55    maennlich  1          2
2 Kontrollgruppe      21    maennlich  3          1
3 Experimentalgruppe  64    weiblich   3          4
4 Experimentalgruppe  19    weiblich   5          4
5 Kontrollgruppe      87    maennlich  1          2
6 Experimentalgruppe  25    maennlich  3          1
7 Kontrollgruppe      63    weiblich   3          4
8 Kontrollgruppe      29    weiblich   5          4
```

Übung 2: Datensatz filtern

Übung 2: Datensatz filtern

Filtern Sie den Datensatz so, dass nur Zeilen mit Befragten übrig bleiben, die eine positive Meinung zu Söder haben und weiblich sind.

Wie viele Fälle umfasst der gefilterte Datensatz?

Lösung für Übung 2: Datensatz filtern

```
41  data %>%
42    filter(mng_soeder >= 4 & geschlecht == "weiblich")
```

Ausgabe in der Console:

```
# A tibble: 4 × 5
  gruppe             alter geschlecht mng_scholz mng_soeder
  <chr>              <chr> <chr>      <chr>      <chr>
1 Experimentalgruppe 64    weiblich   3          4
2 Experimentalgruppe 19    weiblich   5          4
3 Kontrollgruppe     63    weiblich   3          4
4 Kontrollgruppe     29    weiblich   5          4
```

→ Vier der Fälle sind weiblich und haben eine positive Meinung zu Söder.

Variablen erstellen oder Verändern:

`mutate()`

Variablen erstellen oder verändern mit *mutate()*

Struktur:

```
data <- data %>%
    mutate(neuer_Variablenname = *neuer, konstanter Wert*)
```

Oder auch:

```
data <- data %>%
    mutate(neuer_Variablenname = *Transformation bestehender Variable*)
```

Mit **`mutate()`** können

1) neue Variablen hinzugefügt werden.
2) bestehende Variablen verändert werden, indem sie ‚überschrieben' werden. (nicht zu empfehlen)

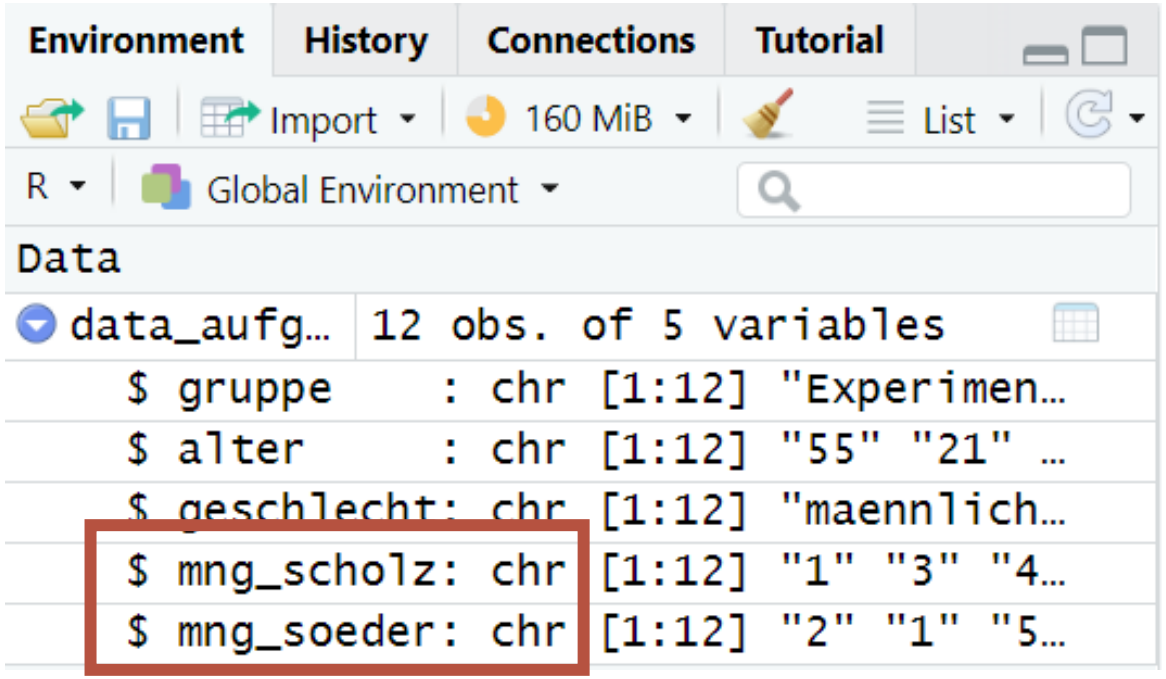
Achtung: Wenn Sie die neu angelegte Variable in weiteren Rechnungen benutzen möchten (meist der Fall!) müssen Sie den veränderten Datensatz mit der neuen Variable wieder zurück in dasselbe oder ein anderes R-Objekt schreiben (mittels „<-").

`mutate()` (Beispiel)

In data_aufg1a gibt es zwei Variablen zur Meinung zu Politikern:

- mng_scholz
- mng_soeder

Beide sollen nun in einer neuen Variable mng_sum zu einem Summenindex aufaddiert werden. Im Environment-Fenster sehen wir allerdings, dass die darin enthaltenen Zahlen als Zeichenketten / String bzw. Text vorliegen, angezeigt durch „chr“, was für „character“ steht.



`mutate()` (Beispiel - Fortsetzung)

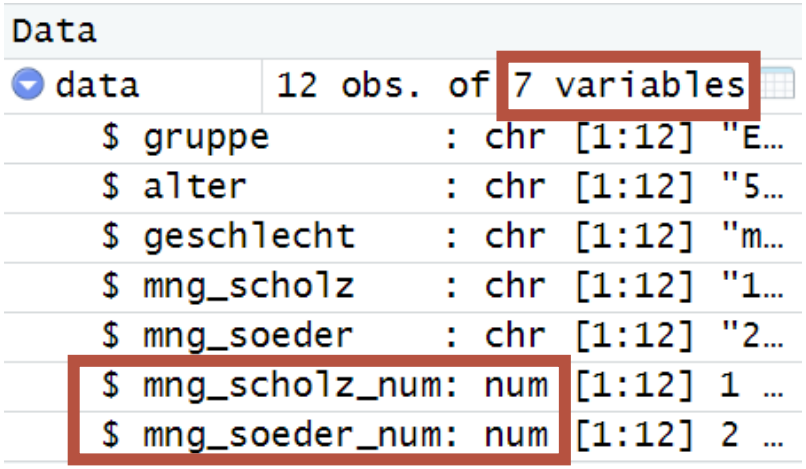
Da wir als Text vorliegende Zahlen nicht in einem Summenindex verrechnen können, müssen wir zunächst zwei neue Variablen

- mng_scholz_num
- mng_soeder_num anlegen, die tatsächlich numerische Werte enthalten.

Zur Erzeugung dieser beiden Variablen können wir `mutate()` mit `as.numeric()` verwenden:

```
45  data <- data %>%
46    mutate(mng_scholz_num = as.numeric(mng_scholz),
47           mng_soeder_num = as.numeric(mng_soeder)
48           )
```

Im Environment- Fenster erscheinen nun die beiden numerischen Variablen:



`mutate()` (Beispiel - Fortsetzung)

Nun können wir den Summenindex mit den beiden neuen Variablen mittels `mutate()` berechnen. Um zu prüfen, ob die Berechnung des Summenindex geklappt hat, lassen wir uns zusätzlich mithilfe der `select()`-Funktion nur die (jetzt fünf) Meinungs-Variablen ausgeben.

```
50  data %>%
51    mutate(mng_sum = (mng_scholz_num + mng_soeder_num)) %>%
52    select(starts_with("mng"))
```

`mutate()` (Beispiel - Fortsetzung)

Ausgabe in der Console:

```
> data %>%
+   mutate(mng_sum = (mng_scholz_num + mng_soeder_num)) %>%
+   select(starts_with("mng"))
# A tibble: 12 × 5
   mng_scholz mng_soeder mng_scholz_num mng_soeder_num mng_sum
   <chr>      <chr>                <dbl>          <dbl>   <dbl>
 1 1          2                        1              2       3
 2 3          1                        3              1       4
 3 4          5                        4              5       9
 4 3          4                        3              4       7
 5 5          4                        5              4       9
 6 2          2                        2              2       4
 7 1          2                        1              2       3
 8 3          1                        3              1       4
 9 4          5                        4              5       9
10 3          4                        3              4       7
11 5          4                        5              4       9
12 2          2                        2              2       4
```

Übung 3: Neue Variable erstellen

Übung 3: Neue Variable erstellen

Ihre Aufgabe lautet nun, die Meinungen über die Politiker Söder und Scholz relativ zu einem (fiktiven!) Durchschnittswert von 3,46 für deutsche Politiker zu berechnen. Fügen Sie zwei neue Variablen in den Datensatz aufg1a_data ein:

- mng_soeder_diff: enthält die Differenz zwischen der Meinung über Söder (mng_soeder_num) und dem Durchschnittswert (3,46)
- mng_scholz_diff: enthält die Differenz zwischen der Meinung über Scholz (mng_scholz_num) und dem Durchschnittswert (3,46)

Fügen Sie diese neuen Variablen dem Datensatz hinzu und lassen Sie sich dann mittels `select()` anzeigen, um zu prüfen, ob die Datentransformation funktioniert hat.

Lösung für Übung 3: Neue Variable erstellen

```
55  data %>%
56    mutate(mng_scholz_diff = mng_scholz_num - 3.46,
57           mng_soeder_diff = mng_soeder_num - 3.46) %>%
58    select(ends_with("_diff"))
```

Ausgabe in der Console:

```
# A tibble: 12 × 2
   mng_scholz_diff mng_soeder_diff
             <dbl>           <dbl>
 1           -2.46           -1.46
 2           -0.46           -2.46
 3            0.54            1.54
 4           -0.46            0.54
 5            1.54            0.54
 6           -1.46           -1.46
 7           -2.46           -1.46
 8           -0.46           -2.46
 9            0.54            1.54
10           -0.46            0.54
11            1.54            0.54
12           -1.46           -1.46
```

Wichtige Take-Aways

- `select()`: Wählt Spalten aus einem Datensatz aus. Kann Spalten nach Namen oder nach einem bestimmten Muster (z.B. mit `starts_with, ends_with, contains`) auswählen.
- `filter()`: Filtert Zeilen basierend auf angegebenen Bedingungen. Kann mehrere Bedingungen kombinieren (logisches UND: `&`, logisches ODER: `|`).
- `mutate()`: Fügt neue Spalten hinzu oder verändert bestehende Spalten.