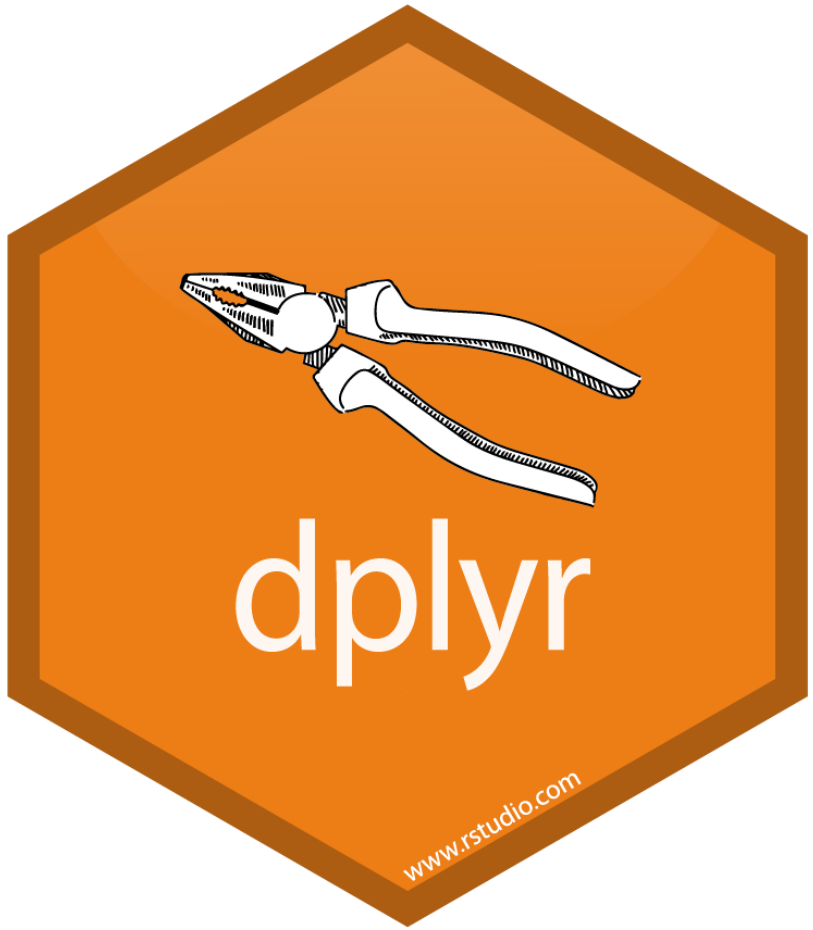
Die Datenaufbereitung mit dplyr

Data Wrangling



Das `dplyr` Paket

Mit `dyplr` haben wir innerhalb des `tidyverse` Funktionalitäten an der Hand, mit denen wir Datensätze in R bearbeiten können:

- Kombination mehrere Operationen durch die *Pipe*: `|>`
  - In alten R-Skripten sehen Sie oft noch die Magritter-Pipe `%>%`, ein Vorläufer der aktuellen *Pipe*
- Datensätze verändern (z.B. neue Spalte) mit `mutate`
- Datensätze nach bestimmten Variablen sortieren mit `arrange`
- Einzelne Reihen herausfiltern mit `filter`
- Einzelne Spalten herauspicken mit `select`

Wir konzentrieren uns wieder auf die heruntergeladenen `gapminder` Daten, welche wir bereits in R importiert und in das richtige Format gebracht haben (`tidy`).

Hier nutzen wir den Datensatz `tidy_data_extended` und laden diesen als `gapminder`:

```
gapminder <- readRDS("data/gapminder_life.rds")
```

Base R versus `tidyverse`

- Durchschnittliche Lebenserwartung weltweit im Jahr 1952 in unserem Datensatz:

Base R versus `tidyverse`

- Durchschnittliche Lebenserwartung weltweit im Jahr 1952 in unserem Datensatz:

Base R

```
avg_1952 <- gapminder[gapminder$jahr == 1952, ]
mean(avg_1952$life_expectancy)
```

```
[1] 51.0925
```

Tidyverse

```
library(dplyr)
gapminder |>
  filter( jahr == 1952 ) |>
  summarize(avg_lifeExp_1952 = mean(life_expectancy)) |>
  pull()
```

```
[1] 51.0925
```

Die Pipe |>

Sie können eine Serie von Befehlen miteinander verknüpfen:

```
gapminder |>
  filter( jahr == 1952 ) |>
  summarize(avg_lifeExp_1952 = mean(life_expectancy))
```

Die Pipe |>

Sie können eine Serie von Befehlen miteinander verknüpfen:

```
gapminder |> # Zuerst der Datensatz und dann ...
  filter( jahr == 1952 ) |>
  summarize(avg_lifeExp_1952 = mean(life_expectancy))
```

Die Pipe |>

Sie können eine Serie von Befehlen miteinander verknüpfen:

```
gapminder |> # Zuerst der Datensatz und dann ...
  filter( jahr == 1952 ) |> # nur das Jahr 1952 und dann ...
  summarize(avg_lifeExp_1952 = mean(life_expectancy))
```

Die Pipe |>

Sie können eine Serie von Befehlen miteinander verknüpfen:

```
gapminder |> # Zuerst der Datensatz und dann ...
  filter( jahr == 1952 ) |> # nur das Jahr 1952 und dann ...
  summarize(avg_lifeExp_1952 = mean(life_expectancy)) # die durchschnittliche Lebenserwartung be
```

```
# A tibble: 1 × 1
  avg_lifeExp_1952
             <dbl>
1             51.1
```

Die sieben wichtigsten Befehle für das data wrangling

- `mutate()`
- `select()`
- `filter()`
- `summarize()`
- `group_by()`
- `arrange()`
- `lead()` und `lag()`

`mutate()`

- Mittels `mutate()` können Sie:
  - eine neue Variable mit einem bestimmten Wert generieren ODER
  - eine neue Variable auf der Basis anderer Variablen generieren ODER
  - den Inhalt einer bestehenden Variablen verändern

mutate()

- Mittels `mutate()` können Sie:
  - eine neue Variable mit einem bestimmten Wert generieren ODER
  - eine neue Variable auf der Basis anderer Variablen generieren ODER
  - den Inhalt einer bestehenden Variablen verändern

```
gapminder_new <- gapminder |>
  mutate(eins = 1)

head(gapminder_new, 4)
```

```
# A tibble: 4 × 5
  country  jahr life_expectancy fertility  eins
  <chr>   <int>           <dbl>     <dbl> <dbl>
1 Brazil   1950            50.3      6.18     1
2 Brazil   1951            50.6      6.17     1
3 Brazil   1952            51.1      6.15     1
4 Brazil   1953            51.6      6.14     1
```

`mutate()`

- Mittels `mutate()` können Sie:
  - eine neue Variable mit einem bestimmten Wert generieren ODER
  - eine neue Variable auf der Basis anderer Variablen generieren ODER
  - den Inhalt einer bestehenden Variablen verändern

`mutate()`

- Mittels `mutate()` können Sie:
  - eine neue Variable mit einem bestimmten Wert generieren ODER
  - eine neue Variable auf der Basis anderer Variablen generieren ODER
  - den Inhalt einer bestehenden Variablen verändern

```
gapminder_new <- gapminder |>
  mutate( life_fertility = life_expectancy * fertility )

head(gapminder_new, 4)
```

```
# A tibble: 4 × 5
  country  jahr life_expectancy fertility life_fertility
  <chr>   <int>           <dbl>     <dbl>          <dbl>
1 Brazil   1950            50.3      6.18           311.
2 Brazil   1951            50.6      6.17           312.
3 Brazil   1952            51.1      6.15           314.
4 Brazil   1953            51.6      6.14           317.
```

`mutate()`

- Mittels `mutate()` können Sie:
  - eine neue Variable mit einem bestimmten Wert generieren ODER
  - eine neue Variable auf der Basis anderer Variablen generieren ODER
  - den Inhalt einer bestehenden Variablen verändern

mutate()

- Mittels `mutate()` können Sie:
  - eine neue Variable mit einem bestimmten Wert generieren ODER
  - eine neue Variable auf der Basis anderer Variablen generieren ODER
  - den Inhalt einer bestehenden Variablen verändern

```
gapminder_new <- gapminder |>
  mutate( life_expectancy = life_expectancy + 5 )

head(gapminder_new, 4)
```

```
# A tibble: 4 × 4
  country  jahr life_expectancy fertility
  <chr>   <int>           <dbl>     <dbl>
1 Brazil   1950            55.3      6.18
2 Brazil   1951            55.6      6.17
3 Brazil   1952            56.1      6.15
4 Brazil   1953            56.6      6.14
```

`select()`

Mittels `select()` können Sie:

- einzelne Variablen aus dem Datensatz selektieren ODER
- einzelne Variablen aus dem Datensatz entfernen

`select()`

Mittels `select()` können Sie:

- einzelne Variablen aus dem Datensatz selektieren ODER
- einzelne Variablen aus dem Datensatz entfernen

```
gapminder_select <- gapminder |>
  select(country, jahr, fertility)

head(gapminder_select, 4)
```

```
# A tibble: 4 × 3
  country  jahr fertility
  <chr>   <int>     <dbl>
1 Brazil   1950      6.18
2 Brazil   1951      6.17
3 Brazil   1952      6.15
4 Brazil   1953      6.14
```

`select()`

Mittels `select()` können Sie:

- einzelne Variablen aus dem Datensatz selektieren ODER
- einzelne Variablen aus dem Datensatz entfernen (mit `-`)

`select()`

Mittels `select()` können Sie:

- ✚ einzelne Variablen aus dem Datensatz selektieren ODER
- ✚ einzelne Variablen aus dem Datensatz entfernen (mit `-`)

```
gapminder_select2 <- gapminder |>
  select(-c(country, fertility))

head(gapminder_select2, 4)
```

```
# A tibble: 4 × 2
   jahr life_expectancy
  <int>           <dbl>
1  1950            50.3
2  1951            50.6
3  1952            51.1
4  1953            51.6
```

`filter()`

Wenn Sie nur die Jahre 2000 und größer betrachten möchten nutzen Sie `filter()`:

```
gapminder_filter <- gapminder |>
  filter(jahr >= 2000)

head(gapminder_filter,4)
```

```
# A tibble: 4 × 4
  country  jahr life_expectancy fertility
  <chr>   <int>           <dbl>     <dbl>
1 Brazil   2000            70.7      2.36
2 Brazil   2001            71.1      2.32
3 Brazil   2002            71.4      2.26
4 Brazil   2003            71.7      2.2
```

`filter()`

Wenn Sie nur die Jahre 2000 und größer betrachten möchten nutzen Sie `filter()`:

```
gapminder_filter <- gapminder |>
  filter(jahr >= 2000)

head(gapminder_filter,4)
```

```
# A tibble: 4 × 4
  country  jahr life_expectancy fertility
  <chr>   <int>           <dbl>     <dbl>
1 Brazil   2000            70.7      2.36
2 Brazil   2001            71.1      2.32
3 Brazil   2002            71.4      2.26
4 Brazil   2003            71.7      2.2
```

- Mit relationalen Operatoren (bspw. >=) können Sie genau spezifizieren, welche Reihen sie herausfiltern möchten.

Relationale Operatoren mit `filter()`

Mit `|` können Sie prüfen, ob *einer* der Filter wahr ist:

Relationale Operatoren mit `filter()`

Mit | können Sie prüfen, ob *einer* der Filter wahr ist:

```
gapminder |>
  filter(jahr >= 2000 | country == "Germany") |>
  head(4)
```

```
# A tibble: 4 × 4
  country  jahr life_expectancy fertility
  <chr>   <int>           <dbl>     <dbl>
1 Brazil   2000            70.7      2.36
2 Brazil   2001            71.1      2.32
3 Brazil   2002            71.4      2.26
4 Brazil   2003            71.7      2.2
```

Relationale Operatoren mit `filter()`

Mit `,` können Sie prüfen, ob *alle* Filter wahr sind:

Relationale Operatoren mit `filter()`

Mit `,` können Sie prüfen, ob *alle* Filter wahr sind:

```
gapminder |>
  filter(jahr >= 2000 , country == "Germany") |>
  head(4)
```

```
# A tibble: 4 × 4
  country  jahr life_expectancy fertility
  <chr>   <int>           <dbl>     <dbl>
1 Germany  2000            78.1      1.35
2 Germany  2001            78.4      1.35
3 Germany  2002            78.6      1.35
4 Germany  2003            78.8      1.35
```

`summarize()`

- `summarize` und `summarise` sind Synonyme und führen beide zum gleichen Ergebnis
- Berechnung von Zusammenfassungen, z.B. die durchschnittliche Lebenserwartung über alle Länder und Jahre hinweg:

summarize()

- `summarize` und `summarise` sind Synonyme und führen beide zum gleichen Ergebnis
- Berechnung von Zusammenfassungen, z.B. die durchschnittliche Lebenserwartung über alle Länder und Jahre hinweg:

```
lifeExp.avg <- gapminder |>
  summarize(durchschnitt = mean(life_expectancy),
            abweichung = sd(life_expectancy))

lifeExp.avg
```

```
# A tibble: 1 × 2
  durchschnitt abweichung
         <dbl>      <dbl>
1         64.6       10.7
```

`summarize()` mit `group_by` kombinieren

Das gleiche Ergebnis für verschiedene Untergruppen erstellen mit `group_by()`:

`summarize()` mit `group_by` kombinieren

Das gleiche Ergebnis für verschiedene Untergruppen erstellen mit `group_by()`:

```
gapminder |>
  group_by( country ) |>
  summarize(durchschnitt = mean(life_expectancy),
            abweichung = sd(life_expectancy))
```

`summarize()` mit `group_by` kombinieren

Das gleiche Ergebnis für verschiedene Untergruppen erstellen mit `group_by()`:

```
gapminder |>
  group_by( country ) |>
  summarize(durchschnitt = mean(life_expectancy),
            abweichung = sd(life_expectancy))
```

Vergessen Sie nicht danach wieder `ungroup()` zu verwenden!

`summarize()` mit `group_by` kombinieren

Das gleiche Ergebnis für verschiedene Untergruppen erstellen mit `group_by()`:

```
gapminder |>
  group_by( country ) |>
  summarize(durchschnitt = mean(life_expectancy),
            abweichung = sd(life_expectancy))
```

Vergessen Sie nicht danach wieder `ungroup()` zu verwenden!

```
grouped_gap <- gapminder |>
  group_by( country ) |>
  summarize(durchschnitt = mean(life_expectancy),
            abweichung = sd(life_expectancy)) |>
  ungroup()

head(grouped_gap, 3)
```

```
# A tibble: 3 × 3
  country durchschnitt abweichung
  <chr>          <dbl>      <dbl>
1 Brazil          64.0       7.29
2 Canada          75.5       4.06
3 China           61.6      11.3
```

`summarize()` mit `.by()` kombinieren

Eine Alternative zu `group_by()` ist der `.by()` Operator (hier benötigen Sie kein `ungroup()` zum Abschluss des Befehls):

`summarize()` mit `.by()` kombinieren

Eine Alternative zu `group_by()` ist der `.by()` Operator (hier benötigen Sie kein `ungroup()` zum Abschluss des Befehls):

```
gapminder |>
  summarize(durchschnitt = mean(life_expectancy),
            abweichung = sd(life_expectancy), .by = country)
```

`summarize()` mit `.by()` kombinieren

Eine Alternative zu `group_by()` ist der `.by()` Operator (hier benötigen Sie kein `ungroup()` zum Abschluss des Befehls):

```
gapminder |>
  summarize(durchschnitt = mean(life_expectancy),
            abweichung = sd(life_expectancy), .by = country)
```

```
grouped_gap_new <- gapminder |>
  summarize(durchschnitt = mean(life_expectancy),
            abweichung = sd(life_expectancy), .by = country)

head(grouped_gap_new, 3)
```

```
# A tibble: 3 × 3
  country durchschnitt abweichung
  <chr>          <dbl>      <dbl>
1 Brazil          64.0       7.29
2 Canada          75.5       4.06
3 China           61.6      11.3
```

`summarize()`

Daten werden durch `summarize()` als `data.frame` abgespeichert.

Zahl als `double` extrahieren:

```
lifeExp.avg |>
  pull(durchschnitt)
```

```
[1] 64.58241
```

`arrange()`

Ordnet den Datensatz nach einer Variablen

- Effizienter als der `order` Befehl aus Base R
- **Default:** Aufsteigende Sortierung
- Möglichkeit: Absteigende Sortierung mit `arrange(desc(fertility))`

`arrange()`

Ordnet den Datensatz nach einer Variablen

- Effizienter als der `order` Befehl aus Base R
- **Default:** Aufsteigende Sortierung
- Möglichkeit: Absteigende Sortierung mit `arrange(desc(fertility))`

```
gapminder |>
  filter( jahr == 1952 ) |>
  arrange( desc(fertility) ) |>
  head(4)
```

```
# A tibble: 4 × 4
  country       jahr life_expectancy fertility
  <chr>        <int>           <dbl>     <dbl>
1 South Africa  1952            44.7      6.31
2 Brazil        1952            51.1      6.15
3 China         1952            42.9      5.97
4 India         1952            35.8      5.9
```

`arrange()`

`arrange()` bietet die Möglichkeit einer verschachtelten Sortierung:

- Zuerst wird nach einer bestimmten Variable sortiert
- Anschließend sortieren wir innerhalb dieser Variable auf eine weitere Variable

> Sortieren Sie erst nach `life_expectancy` und anschließend nach `fertility`

`arrange()`

`arrange()` bietet die Möglichkeit einer verschachtelten Sortierung:

- Zuerst wird nach einer bestimmten Variable sortiert
- Anschließend sortieren wir innerhalb dieser Variable auf eine weitere Variable

Sortieren Sie erst nach `life_expectancy` und anschließend nach `fertility`

```
gapminder |>
  filter( jahr == 1952 ) |>
  arrange( desc(life_expectancy), desc(fertility) ) |>
  head(4)
```

```
# A tibble: 4 × 4
  country  jahr life_expectancy fertility
  <chr>   <int>           <dbl>     <dbl>
1 Canada   1952            68.7      3.62
2 Germany  1952            67.4      2.11
3 Russia   1952            58.2      2.88
4 Brazil   1952            51.1      6.15
```

`lead()` und `lag()`

- `lead()` liest den vorausgehende Werte einer Variablen aus
- `lag()` liest den darauffolgenden Werte einer Variablen aus
- Insbesondere bei der Arbeit mit Zeitreihendaten wichtig
  - Kann gut mit `group_by` kombiniert werden

`lead()` und `lag()`

- `lead()` liest den vorausgehende Werte einer Variablen aus
- `lag()` liest den darauffolgenden Werte einer Variablen aus
- Insbesondere bei der Arbeit mit Zeitreihendaten wichtig
  - Kann gut mit `group_by` kombiniert werden

```
gapminder |>
  arrange( country, jahr ) |>
  mutate(lag_fertility = lag(fertility),
         lead_fertility = lead(fertility)) |>
  select(country, jahr, fertility, lag_fertility, lead_fertility) |>
  head(4)
```

`lead()` und `lag()`

- `lead()` liest den vorausgehende Werte einer Variablen aus
- `lag()` liest den darauffolgenden Werte einer Variablen aus
- Insbesondere bei der Arbeit mit Zeitreihendaten wichtig
  - Kann gut mit `group_by` kombiniert werden

```
gapminder |>
  arrange( country, jahr ) |>
  mutate(lag_fertility = lag(fertility),
         lead_fertility = lead(fertility)) |>
  select(country, jahr, fertility, lag_fertility, lead_fertility) |>
  head(4)
```

```
# A tibble: 4 × 5
  country  jahr fertility lag_fertility lead_fertility
  <chr>   <int>     <dbl>          <dbl>          <dbl>
1 Brazil   1950      6.18          NA              6.17
2 Brazil   1951      6.17           6.18           6.15
3 Brazil   1952      6.15           6.17           6.14
4 Brazil   1953      6.14           6.15           6.14
```