

# Einführung in R

Fortbildung im Institut der deutschen Wirtschaft

---

# 6 Deskriptive Statistiken

# Inhalte und Ziele der Sitzung

- Berechnen deskriptiver Statistiken
- Häufigkeitsstatistiken für kategoriale Variablen

# Deskriptive Statistiken

```
library(tidyverse)
# install.packages(palmerpenguins)
(beispieldaten <- palmerpenguins::penguins)
```

```
#> # A tibble: 344 x 8
#>   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
#>   <fct>   <fct>         <dbl>         <dbl>           <int>         <int> <fct> <int>
#> 1 Adelie Torgersen      39.1           18.7             181           3750 male   2007
#> 2 Adelie Torgersen      39.5           17.4             186           3800 female 2007
#> 3 Adelie Torgersen      40.3           18              195           3250 female 2007
#> 4 Adelie Torgersen      NA              NA              NA           NA <NA>   2007
#> 5 Adelie Torgersen      36.7           19.3             193           3450 female 2007
#> 6 Adelie Torgersen      39.3           20.6             190           3650 male   2007
#> 7 Adelie Torgersen      38.9           17.8             181           3625 female 2007
#> 8 Adelie Torgersen      39.2           19.6             195           4675 male   2007
#> 9 Adelie Torgersen      34.1           18.1             193           3475 <NA>   2007
#> 10 Adelie Torgersen      42            20.2             190           4250 <NA>   2007
#> # ... with 334 more rows
```

# summary()

- Deskriptive Statistiken fassen Variablen zusammen.
- Einen schnellen Überblick über einen Datensatz kann man sich mit der `summary()`-Funktion verschaffen:

```
summary(beispieldaten)
```

```
#>      species      island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
#> Adelie      :152  Biscoe      :168   Min.       :32.10   Min.       :13.10   Min.       :172.0   Min.       :2700
#> Chinstrap: 68   Dream       :124   1st Qu.:39.23   1st Qu.:15.60   1st Qu.:190.0   1st Qu.:3550
#> Gentoo      :124  Torgersen: 52   Median :44.45   Median :17.30   Median :197.0   Median :4050
#>                                     Mean      :43.92   Mean      :17.15   Mean      :200.9   Mean      :4202
#>                                     3rd Qu.:48.50   3rd Qu.:18.70   3rd Qu.:213.0   3rd Qu.:4750
#>                                     Max.       :59.60   Max.       :21.50   Max.       :231.0   Max.       :6300
#>                                     NA's       :2      NA's       :2      NA's       :2      NA's       :2
#>      sex      year
#> female:165   Min.    :2007
#> male   :168   1st Qu.:2007
#> NA's    : 11   Median :2008
#>                                     Mean      :2008
#>                                     3rd Qu.:2009
#>                                     Max.       :2009
#>
```

- Eine ähnliche Funktionsweise bietet die Funktion `skimr::skim()`.
- Diese Funktionen können einen ersten Überblick über die Daten geben, sind jedoch für die weitere Arbeit mit den errechneten Werten nicht hilfreich.
- Deswegen: »Selbst sind die Forschenden.« Die meisten notwendigen Funktionen zur Erstellung deskriptiver Statistiken kennen wir ohnehin bereits.

- Um ausgewählte Summary Statistics für numerische Variablen zu berechnen, nutzen wir die Funktion `summarise()` in Kombination mit der entsprechenden Funktion, die wir anwenden möchten:

Funktion	Beschreibung
<code>mean(x, na.rm = FALSE)</code>	Arithmetic mean
<code>sd(x)</code>	(Sample) Standard Deviation
<code>var(x)</code>	(Sample) Variance
<code>median(x)</code>	Median
<code>quantile(x, probs, type)</code>	Quantile of x. probs: vector with probabilities
<code>min(x)</code>	Minimum value of x
<code>max(x)</code>	Maximum value of x
<code>range(x)</code>	x_min and x_max
...	...



# Eine Variable, eine Funktion

- Häufig sind wir an **einer** Summary Statistic **einer** Variable interessiert. Z. B.:

```
beispieldaten |>
  summarise(durchschnitt_bill_length_mm = mean(bill_length_mm, na.rm = T))
```

```
#> # A tibble: 1 x 1
#>   durchschnitt_bill_length_mm
#>   <dbl>
#> 1           43.9
```

- `na.rm = T`: Falls NA-Werte (fehlende Werte) in der Variable auftauchen, sorgt dieses Argument dafür, dass dennoch die Funktion angewendet wird.

# Mehrere Variablen, mehrere Funktionen

- Wir können in einem `summarise()`-Schritt auch mehrere Funktionen manuell kombinieren:

```
beispieldaten |>
  summarise(durchschnitt_bill_length_mm = mean(bill_length_mm, na.rm = T),
            median_bill_length_mm = median(bill_length_mm, na.rm = T))
```

```
#> # A tibble: 1 x 2
#>   durchschnitt_bill_length_mm median_bill_length_mm
#>   <dbl>                <dbl>
#> 1      43.9                44.4
```

# Viele Variablen, viele Funktionen

- Das »hardcoden« von Variablennamen und Funktionen ist bei wenigen Variablen und Funktionen kein großer Schreibaufwand.
- Bei mehreren Variablen und Funktionen wird das Tippen jedoch schnell zeitintensiv und fehleranfällig.
- Die Funktion `across()` kann hierbei helfen:

```
beispieldaten |>
```

```
  summarise(across(.cols = c("bill_length_mm", "bill_depth_mm"),  
                    .fns = list(durchschnitt = mean, stabw = sd),  
                    na.rm = T  
              )  
            )
```

```
#> # A tibble: 1 x 4
```

```
#>   bill_length_mm_durchschnitt bill_length_mm_stabw bill_depth_mm_durchschnitt bill_depth_mm_stabw  
#>               <dbl>               <dbl>               <dbl>               <dbl>  
#> 1             43.9             5.46             17.2             1.97
```

- Hinweis: `across()` funktioniert auch mit `mutate()` beim Erstellen neuer oder Überschreiben bestehender Variablen.

# Viele Variablen, viele Funktionen

- Richtig Schreibaufwand lässt sich sparen, wenn weitere Hilfsfunktionen verwendet werden.
- So kann man Beispielsweise die Liste der Funktionen auf alle numerischen Spalten anwenden:

```
beispieldaten |>
```

```
  summarise(across(.cols = where(is.numeric),  
                    .fns = list(durchschnitt = mean, stabw = sd),  
                    na.rm = T  
                    )  
            )
```

```
#> # A tibble: 1 x 10
```

```
#>   bill_length_mm_d~ bill_length_mm_s~ bill_depth_mm_du~ bill_depth_mm_s~ flipper_length_~ flipper_len
```

```
#>           <dbl>           <dbl>           <dbl>           <dbl>           <dbl>
```

```
#> 1           43.9           5.46           17.2           1.97           201.
```

```
#> # ... with 4 more variables: body_mass_g_durchschnitt <dbl>, body_mass_g_stabw <dbl>,
```

```
#> #   year_durchschnitt <dbl>, year_stabw <dbl>
```

## Für Gruppen

```
beispieldaten |>
  group_by(year) |>
  summarise(across(.cols = where(is.numeric),
                    .fns = list(durchschnitt = mean, stabw = sd),
                    na.rm = T
                  )
            )
```

```
#> # A tibble: 3 x 9
#>   year bill_length_mm_durchschnitt bill_length_mm_s~ bill_depth_mm_du~ bill_depth_mm_s~ flipper_leng
#>   <int>                <dbl>                <dbl>                <dbl>                <dbl>
#> 1  2007                43.7                5.39                17.4                2.15
#> 2  2008                43.5                5.34                16.9                1.98
#> 3  2009                44.5                5.64                17.1                1.79
#> # ... with 3 more variables: flipper_length_mm_stabw <dbl>, body_mass_g_durchschnitt <dbl>,
#> #   body_mass_g_stabw <dbl>
```

- Kreuztabellen können als gruppenweise Summary Statistics für zwei Gruppen verstanden werden.
- Mit dieser Interpretation ist das Erstellen von Kreuztabellen in der bekannten Syntax möglich, es muss jedoch für zwei (oder auch mehrere) Variablen gruppiert werden:

```
beispieldaten |>  
  group_by(year, species) |>  
  summarise(mean_bill_depth_mm = mean(bill_depth_mm, na.rm = T)) |>  
  ungroup()
```

```
#> # A tibble: 9 x 3  
#>   year species mean_bill_depth_mm  
#>   <int> <fct>          <dbl>  
#> 1  2007 Adelie          18.8  
#> 2  2007 Chinstrap      18.5  
#> 3  2007 Gentoo         14.7  
#> 4  2008 Adelie          18.2  
#> 5  2008 Chinstrap      18.4  
#> 6  2008 Gentoo         14.9  
#> 7  2009 Adelie          18.1  
#> 8  2009 Chinstrap      18.3  
#> 9  2009 Gentoo         15.3
```

- Die so erstellte Tabelle kann mit `pivot_wider()` in das typische Kreuztabellenformat gebracht werden. *Hinweis: Dann genügt die Tabelle nicht mehr dem Tidy Data Prinzip.*

```
beispieldaten |>
  group_by(year, species) |>
  summarise(mean_bill_depth_mm = mean(bill_depth_mm, na.rm = T)) |>
  ungroup() |>
  pivot_wider(id_cols = "year",
              names_from = "species",
              values_from = "mean_bill_depth_mm")
```

```
#> # A tibble: 3 x 4
#>   year Adelie Chinstrap Gentoo
#>   <int> <dbl>    <dbl> <dbl>
#> 1  2007  18.8      18.5  14.7
#> 2  2008  18.2      18.4  14.9
#> 3  2009  18.1      18.3  15.3
```

# Häufigkeiten und Anteile



- Als Teil deskriptiver Statistiken sind wir auch an Häufigkeiten und Anteilen verschiedener Gruppen interessiert.
- Z. B.: Wie viele der beobachteten Pinguine gehören zu welcher Spezies? Wie groß ist der Anteil der beobachteten weiblichen Pinguine nach Spezies und Jahr.
- Die Funktion `count()` zählt die einzigartigen Ausprägungen der übergebenen Variable(n). Bei mehreren Variablen wird paarweise ausgewertet.
- D. h. `count()` übernimmt die Gruppierung für uns und `group_by()` kann entfallen:

```
beispieldaten |>
  group_by(species) |>
  count()
```

```
#> # A tibble: 3 x 2
#> # Groups:   species [3]
#>   species      n
#>   <fct>    <int>
#> 1 Adelie    152
#> 2 Chinstrap  68
#> 3 Gentoo   124
```

```
beispieldaten |>
  count(species)
```

```
#> # A tibble: 3 x 2
#>   species      n
#>   <fct>    <int>
#> 1 Adelie    152
#> 2 Chinstrap  68
#> 3 Gentoo   124
```

# Paarweise Häufigkeitstabellen

- Paarweise Häufigkeiten lassen sich, wie bereits bei den Summary Statistics gezeigt, in Kreuztabellen überführen:

```
beispieldaten |>  
  count(species, year)
```

```
#> # A tibble: 9 x 3  
#>   species    year     n  
#>   <fct>    <int> <int>  
#> 1 Adelie    2007     50  
#> 2 Adelie    2008     50  
#> 3 Adelie    2009     52  
#> 4 Chinstrap 2007     26  
#> 5 Chinstrap 2008     18  
#> 6 Chinstrap 2009     24  
#> 7 Gentoo    2007     34  
#> 8 Gentoo    2008     46  
#> 9 Gentoo    2009     44
```

```
beispieldaten |>  
  count(species, year) |>  
  pivot_wider(id_cols = "year",  
              names_from = "species",  
              values_from = "n")
```

```
#> # A tibble: 3 x 4  
#>   year Adelie Chinstrap Gentoo  
#>   <int> <int>      <int> <int>  
#> 1  2007     50         26     34  
#> 2  2008     50         18     46  
#> 3  2009     52         24     44
```

- Die Berechnung von Anteilen erfolgt schlicht als Anzahl der Ausprägung in Relation zur Summe der Ausprägungen innerhalb der Gruppe, die von Interesse ist. Im einfachsten Fall also als:

```
beispieldaten |>
  count(species) |>
  mutate(anteil = n / sum(n))
```

```
#> # A tibble: 3 x 3
#>   species      n anteil
#>   <fct>    <int> <dbl>
#> 1 Adelie    152  0.442
#> 2 Chinstrap  68  0.198
#> 3 Gentoo   124  0.360
```

- Bei gruppenweisen Anteilen übergeben wir mit `group_by()` die Variable(n) innerhalb derer die Anteile bestimmt werden sollen.
- Zuerst Häufigkeiten berechnen:
- Dann gruppieren und Anteile berechnen:

```
beispieldaten |>
  count(species, year)
```

```
#> # A tibble: 9 x 3
#>   species    year     n
#>   <fct>    <int> <int>
#> 1 Adelie    2007     50
#> 2 Adelie    2008     50
#> 3 Adelie    2009     52
#> 4 Chinstrap 2007     26
#> 5 Chinstrap 2008     18
#> 6 Chinstrap 2009     24
#> 7 Gentoo    2007     34
#> 8 Gentoo    2008     46
#> 9 Gentoo    2009     44
```

```
beispieldaten |>
  count(species, year) |>
  group_by(year) |>
  mutate(anteil = n / sum(n))
```

```
#> # A tibble: 9 x 4
#> # Groups:   year [3]
#>   species    year     n anteil
#>   <fct>    <int> <int> <dbl>
#> 1 Adelie    2007     50  0.455
#> 2 Adelie    2008     50  0.439
#> 3 Adelie    2009     52  0.433
#> 4 Chinstrap 2007     26  0.236
#> 5 Chinstrap 2008     18  0.158
#> 6 Chinstrap 2009     24  0.2
#> 7 Gentoo    2007     34  0.309
#> 8 Gentoo    2008     46  0.404
#> 9 Gentoo    2009     44  0.367
```

- Bei Bedarf kann dann wieder in ein Kreuztabellenformat überführt werden:

```
beispieldaten |>
  count(species, year) |>
  group_by(year) |>
  mutate(anteil = n / sum(n)) |>
  ungroup() |>
  pivot_wider(names_from = "species",
              id_cols = "year",
              values_from = c("n", "anteil"))
```

```
#> # A tibble: 3 x 7
#>   year n_Adelie n_Chinstrap n_Gentoo anteil_Adelie anteil_Chinstrap anteil_Gentoo
#>   <int>   <int>      <int>   <int>      <dbl>         <dbl>         <dbl>
#> 1  2007     50        26     34      0.455         0.236         0.309
#> 2  2008     50        18     46      0.439         0.158         0.404
#> 3  2009     52        24     44      0.433         0.2         0.367
```

# Aus R in die Kommunikation

- Tabellen können in allen bekannten Datenformaten gespeichert werden.
- Beispieldatensatz, der gespeichert werden soll:

```
(daten_speich <- beispieldaten |>  
  count(species))
```

```
#> # A tibble: 3 x 2  
#>   species      n  
#>   <fct>    <int>  
#> 1 Adelie    152  
#> 2 Chinstrap  68  
#> 3 Gentoo   124
```

- Für die Weiterverarbeitung in Reports und anderen IW-Formaten bieten sich die gängigen Excel-Formate, insbesondere xlsx-Dateien, an.

# Funktionen zum Speichern von Tabellen


## Good to know:

Paket	Funktion	Beispiel	Anwendung
Base R	<code>write.table()</code>	<code>write.table(x = daten_speich, file = "daten_speich.txt")</code>	Für rohe Textdateien, Trennzeichen ist Leerzeichen, Dezimaltrennzeichen ist Punkt.
Base R	<code>write.csv()</code>	<code>write.csv(x = daten_speich, file = "daten_speich.csv")</code>	Für »Englische« CSV-Dateien; Trennzeichen ist Komma, Dezimaltrennzeichen ist Punkt.
Base R	<code>write.csv2()</code>	<code>write.csv2(x = daten_speich, file = "daten_speich.csv")</code>	Für »Deutsche« CSV-Dateien; Trennzeichen ist Semikolon, Dezimaltrennzeichen ist Komma.

## Empfohlene Funktionen:

Paket	Funktion	Beispiel	Anwendung
readr	<code>write_csv()</code>	<code>write_csv(x = daten_speich, file = "daten_speich.csv")</code>	»Englische« CSV-Dateien; Trennzeichen ist Komma, Dezimaltrennzeichen ist Punkt. Doppelt so schnell wie <code>write.csv()</code> .
readr	<code>write_csv2()</code>	<code>write_csv2(x = daten_speich, file = "daten_speich.csv")</code>	Für »Deutsche« CSV-Dateien; Trennzeichen ist Semikolon, Dezimaltrennzeichen ist Komma. Doppelt so schnell wie <code>write.csv2()</code> .
xlsx	<code>write.xlsx()</code>	<code>write.xlsx(x = daten_speich, file = "daten_speich.csv", sheetName = "Tabelle1")</code>	Zum Speichern von xlsx-Dateien.



Genug der Theorie. Ab nach  Studio®.