3. Häufigkeitstabellen

Zählen und Auszählen mit janitor::tabyl()

Dr. Paul Schmidt

Um alle in diesem Kapitel verwendeten Pakete zu installieren und zu laden, führt man folgenden Code aus:

```
for (pkg in c("janitor", "scales", "tidyverse")) {
  if (!require(pkg, character.only = TRUE)) install.packages(pkg)
}

library(janitor)
library(scales)
library(tidyverse)
```

Einleitung

Häufigkeitstabellen sind eines der grundlegendsten Werkzeuge der Datenanalyse. Wie oft kommt jede Kategorie vor? Wie verteilen sich die Werte über verschiedene Gruppen? Solche Fragen stellen wir uns ständig – sei es bei der Qualitätskontrolle, bei Umfrageauswertungen oder einfach um einen ersten Überblick über die Daten zu bekommen.

R bietet mehrere Wege, Häufigkeitstabellen zu erstellen. In diesem Kapitel beginnen wir mit den Grundlagen (`table()` und `count()`), um dann zu verstehen, warum `janitor::tabyl()` in den meisten Fällen die elegantere und praktischere Lösung ist.

Beispieldaten

Für dieses Kapitel verwenden wir den `starwars`-Datensatz aus dem {dplyr}-Paket. Er enthält Informationen über 87 Charaktere aus dem Star-Wars-Universum:

```
glimpse(starwars)
```

```
Rows: 87
Columns: 14
$ name       <chr> "Luke Skywalker", "C-3PO", "R2-D2", "Darth Vader", "Leia Or…
$ height     <int> 172, 167, 96, 202, 150, 178, 165, 97, 183, 182, 188, 180, 2…
$ mass       <dbl> 77.0, 75.0, 32.0, 136.0, 49.0, 120.0, 75.0, 32.0, 84.0, 77.…
$ hair_color <chr> "blond", NA, NA, "none", "brown", "brown, grey", "brown", N…
$ skin_color <chr> "fair", "gold", "white, blue", "white", "light", "light", "…
$ eye_color  <chr> "blue", "yellow", "red", "yellow", "brown", "blue", "blue",…
$ birth_year <dbl> 19.0, 112.0, 33.0, 41.9, 19.0, 52.0, 47.0, NA, 24.0, 57.0, …
$ sex        <chr> "male", "none", "none", "male", "female", "male", "female",…
$ gender     <chr> "masculine", "masculine", "masculine", "masculine", "femini…
$ homeworld  <chr> "Tatooine", "Tatooine", "Naboo", "Tatooine", "Alderaan", "T…
$ species    <chr> "Human", "Droid", "Droid", "Human", "Human", "Human", "Huma…
$ films      <list> <"A New Hope", "The Empire Strikes Back", "Return of the J…
$ vehicles   <list> <"Snowspeeder", "Imperial Speeder Bike">, <>, <>, <>, "Imp…
$ starships  <list> <"X-wing", "Imperial shuttle">, <>, <>, "TIE Advanced x1",…
```

Der Datensatz hat sowohl kategoriale Variablen (wie `species`, `sex`, `homeworld`) als auch numerische Variablen (wie `height`, `mass`). Für die meisten Beispiele filtern wir auf Menschen (`species == "Human"`), um die Ausgaben übersichtlicher zu halten:

```
humans <- starwars %>%
  filter(species == "Human")

humans
```

```
# A tibble: 35 × 14
   name     height  mass hair_color skin_color eye_color birth_year sex   gender
   <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
 1 Luke Sk…    172    77 blond      fair       blue            19   male  mascu…
 2 Darth V…    202   136 none       white      yellow          41.9 male  mascu…
 3 Leia Or…    150    49 brown      light      brown           19   fema… femin…
 4 Owen La…    178   120 brown, gr… light      blue            52   male  mascu…
 5 Beru Wh…    165    75 brown      light      blue            47   fema… femin…
 6 Biggs D…    183    84 black      light      brown           24   male  mascu…
 7 Obi-Wan…    182    77 auburn, w… fair       blue-gray       57   male  mascu…
 8 Anakin …    188    84 blond      fair       blue            41.9 male  mascu…
 9 Wilhuff…    180    NA auburn, g… fair       blue            64   male  mascu…
10 Han Solo    180    80 brown      fair       brown           29   male  mascu…
# i 25 more rows
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

Der klassische Weg: table()

Die Funktion `table()` ist in base R eingebaut und erstellt einfache Häufigkeitstabellen:

```
table(humans$eye_color)
```

```
     blue blue-gray     brown      dark     hazel   unknown    yellow 
       12         1        16         1         2         1         2 
```

Das funktioniert, hat aber einige Nachteile:

1. **Kein data.frame**: Das Ergebnis ist ein `table`-Objekt, kein Tibble/data.frame. Es lässt sich nicht direkt mit tidyverse-Funktionen weiterverarbeiten.
2. **Keine Prozente**: Wir erhalten nur absolute Zahlen, keine relativen Häufigkeiten.
3. **Umständliche Syntax**: Bei mehreren Variablen wird es schnell unübersichtlich.

Man kann das Ergebnis zwar in einen data.frame umwandeln, aber das ist umständlich:

```
table(humans$eye_color) %>%
  as.data.frame()
```

```
       Var1 Freq
1      blue   12
2 blue-gray    1
3     brown   16
4      dark    1
5     hazel    2
6   unknown    1
7    yellow    2
```

Die Spaltennamen sind nicht intuitiv (`Var1`, `Freq`), und Prozente müssen wir selbst berechnen.

Der tidyverse-Weg: count() + mutate()

Mit `dplyr::count()` bekommen wir direkt einen Tibble zurück:

```
humans %>%
  count(eye_color)
```

```
# A tibble: 7 × 2
  eye_color     n
  <chr>     <int>
1 blue         12
2 blue-gray     1
3 brown        16
4 dark          1
5 hazel         2
6 unknown       1
7 yellow        2
```

Das ist schon besser! Wenn wir Prozente möchten, fügen wir sie mit `mutate()` hinzu:

```
humans %>%
  count(eye_color) %>%
  mutate(
    percent = n / sum(n),
```

```
    percent_formatted = percent(percent, accuracy = 0.1)
  )
```

```
# A tibble: 7 × 4
  eye_color     n percent percent_formatted
  <chr>     <int>   <dbl> <chr>
1 blue         12  0.343  34.3%
2 blue-gray     1  0.0286 2.9%
3 brown        16  0.457  45.7%
4 dark          1  0.0286 2.9%
5 hazel         2  0.0571 5.7%
6 unknown       1  0.0286 2.9%
7 yellow        2  0.0571 5.7%
```

Und wenn wir eine Summenzeile möchten, müssen wir diese separat berechnen und dann anhängen:

```
# Schritt 1: Häufigkeiten pro Kategorie berechnen
pro_eye_color <- humans %>%
  count(eye_color) %>%
  mutate(percent = n / sum(n))

pro_eye_color
```

```
# A tibble: 7 × 3
  eye_color     n percent
  <chr>     <int>   <dbl>
1 blue         12  0.343
2 blue-gray     1  0.0286
3 brown        16  0.457
4 dark          1  0.0286
5 hazel         2  0.0571
6 unknown       1  0.0286
7 yellow        2  0.0571
```

```
# Schritt 2: Gesamtzeile separat erstellen
gesamt <- tibble(
  eye_color = "Total",
  n = sum(pro_eye_color$n),
  percent = 1
)

gesamt
```

```
# A tibble: 1 × 3
  eye_color     n percent
  <chr>     <int>   <dbl>
1 Total        35       1
```

```
# Schritt 3: Zusammenfügen
bind_rows(pro_eye_color, gesamt)
```

```
# A tibble: 8 × 3
  eye_color     n percent
  <chr>     <int>   <dbl>
1 blue         12  0.343
2 blue-gray     1  0.0286
3 brown        16  0.457
4 dark          1  0.0286
5 hazel         2  0.0571
6 unknown       1  0.0286
7 yellow        2  0.0571
8 Total        35  1
```

Das funktioniert, aber es ist viel Tipparbeit für eine so häufige Aufgabe. Hier kommt `tabyl()` ins Spiel.

janitor::tabyl() – Die elegante Lösung

Die Funktion `tabyl()` aus dem {janitor}-Paket wurde genau für diesen Anwendungsfall entwickelt. Sie kombiniert die besten Eigenschaften von `table()` und `count()` und fügt weitere nützliche Features hinzu.

Einweg-Tabelle (eine Variable)

```
humans %>%
  tabyl(eye_color)
```

```
 eye_color  n    percent
      blue 12 0.34285714
 blue-gray  1 0.02857143
     brown 16 0.45714286
      dark  1 0.02857143
     hazel  2 0.05714286
   unknown  1 0.02857143
    yellow  2 0.05714286
```

Mit einem einzigen Funktionsaufruf erhalten wir:

- **n**: Die absolute Häufigkeit
- **percent**: Den relativen Anteil (als Dezimalzahl)

Das Ergebnis ist ein Tibble, den wir direkt weiterverarbeiten können.

NA-Werte kontrollieren

Schauen wir uns eine Variable mit fehlenden Werten an – `homeworld` enthält mehrere NA-Einträge:

```
humans %>%
  tabyl(homeworld)
```

```
    homeworld n    percent valid_percent
     Alderaan 3 0.08571429    0.10344828
       Bespin 1 0.02857143    0.03448276
    Chandrila 1 0.02857143    0.03448276
 Concord Dawn 1 0.02857143    0.03448276
     Corellia 2 0.05714286    0.06896552
    Coruscant 2 0.05714286    0.06896552
       Eriadu 1 0.02857143    0.03448276
   Haruun Kal 1 0.02857143    0.03448276
       Kamino 1 0.02857143    0.03448276
        Naboo 5 0.14285714    0.17241379
      Serenno 1 0.02857143    0.03448276
      Socorro 1 0.02857143    0.03448276
      Stewjon 1 0.02857143    0.03448276
     Tatooine 8 0.22857143    0.27586207
         <NA> 6 0.17142857            NA
```

Standardmäßig zeigt `tabyl()` NA-Werte als eigene Kategorie an. Beachte die zwei Prozent-Spalten:

- **percent**: Anteil bezogen auf alle Zeilen (inkl. NA)
- **valid_percent**: Anteil bezogen auf gültige Werte (ohne NA)

Mit `show_na = FALSE` können wir NA-Werte ausblenden:

```
humans %>%
  tabyl(homeworld, show_na = FALSE)
```

```
    homeworld n    percent
     Alderaan 3 0.10344828
       Bespin 1 0.03448276
    Chandrila 1 0.03448276
 Concord Dawn 1 0.03448276
     Corellia 2 0.06896552
    Coruscant 2 0.06896552
       Eriadu 1 0.03448276
   Haruun Kal 1 0.03448276
       Kamino 1 0.03448276
        Naboo 5 0.17241379
      Serenno 1 0.03448276
      Socorro 1 0.03448276
      Stewjon 1 0.03448276
     Tatooine 8 0.27586207
```

Wenn wir `show_na = FALSE` setzen, gibt es nur noch eine Prozent-Spalte, da beide Werte identisch wären.

Leere Kategorien anzeigen

Wenn eine Variable als Faktor definiert ist, kann es Levels geben, die im Datensatz nicht vorkommen. Mit `show_missing_levels = TRUE` werden diese trotzdem angezeigt:

```
# Beispiel: Faktor mit Level, das nicht vorkommt
humans_factor <- humans %>%
  mutate(eye_color = factor(eye_color,
                            levels = c("blue", "brown", "hazel", "dark", "green",
"blue-gray")))

humans_factor %>%
  tabyl(eye_color, show_missing_levels = TRUE)
```

```
 eye_color  n    percent valid_percent
      blue 12 0.34285714       0.37500
     brown 16 0.45714286       0.50000
     hazel  2 0.05714286       0.06250
      dark  1 0.02857143       0.03125
     green  0 0.00000000       0.00000
 blue-gray  1 0.02857143       0.03125
      <NA>  3 0.08571429            NA
```

Das Level “green” kommt bei Menschen nicht vor, wird aber trotzdem mit n=0 angezeigt. Das ist besonders nützlich bei Umfragedaten, wo bestimmte Antwortkategorien möglicherweise von niemandem gewählt wurden, aber trotzdem im Bericht erscheinen sollen.

> Übung: Einweg-Tabellen
>
> Erstelle mit dem `humans`-Datensatz folgende Tabellen:
>
> **a)** Eine Häufigkeitstabelle für die Variable `gender`.
>
> **b)** Eine Häufigkeitstabelle für `homeworld`, bei der NA-Werte ausgeblendet sind.

Lösungsvorschlag

```
# a) Geschlechterverteilung
humans %>%
 tabyl(gender)
```

```
    gender  n   percent
  feminine  9 0.2571429
 masculine 26 0.7428571
```

```
# b) Heimatwelten ohne NA
humans %>%
 tabyl(homeworld, show_na = FALSE)
```

```
    homeworld n    percent
     Alderaan 3 0.10344828
       Bespin 1 0.03448276
    Chandrila 1 0.03448276
 Concord Dawn 1 0.03448276
     Corellia 2 0.06896552
    Coruscant 2 0.06896552
       Eriadu 1 0.03448276
   Haruun Kal 1 0.03448276
       Kamino 1 0.03448276
        Naboo 5 0.17241379
      Serenno 1 0.03448276
      Socorro 1 0.03448276
      Stewjon 1 0.03448276
     Tatooine 8 0.27586207
```

Zweiweg-Tabellen (Kreuztabellen)

Mit zwei Variablen erstellt `tabyl()` automatisch eine Kreuztabelle:

```
humans %>%
  tabyl(eye_color, gender)
```

```
 eye_color feminine masculine
      blue         3          9
 blue-gray         0          1
     brown         4         12
      dark         0          1
     hazel         1          1
   unknown         1          0
    yellow         0          2
```

Die erste Variable (`eye_color`) definiert die Zeilen, die zweite (`gender`) die Spalten. Das Ergebnis zeigt die absoluten Häufigkeiten für jede Kombination.

Dreiweg-Tabellen

Mit drei Variablen erstellt `tabyl()` eine Liste von Kreuztabellen – eine für jede Ausprägung der dritten Variable:

```
humans %>%
  tabyl(eye_color, gender, hair_color)
```

```
$auburn
 eye_color feminine masculine
      blue         1          0
 blue-gray         0          0
     brown         0          0
      dark         0          0
     hazel         0          0
   unknown         0          0
    yellow         0          0

$`auburn, grey`
 eye_color feminine masculine
      blue         0          1
 blue-gray         0          0
     brown         0          0
      dark         0          0
     hazel         0          0
   unknown         0          0
    yellow         0          0

$`auburn, white`
 eye_color feminine masculine
      blue         0          0
 blue-gray         0          1
     brown         0          0
      dark         0          0
     hazel         0          0
   unknown         0          0
    yellow         0          0

$black
 eye_color feminine masculine
      blue         0          0
 blue-gray         0          0
     brown         1          6
      dark         0          1
```

```
     hazel         0          0
   unknown         0          0
    yellow         0          0

$blond
 eye_color feminine masculine
      blue         0          3
 blue-gray         0          0
     brown         0          0
      dark         0          0
     hazel         0          0
   unknown         0          0
    yellow         0          0

$brown
 eye_color feminine masculine
      blue         1          3
 blue-gray         0          0
     brown         3          4
      dark         0          0
     hazel         1          1
   unknown         0          0
    yellow         0          0

$`brown, grey`
 eye_color feminine masculine
      blue         0          1
 blue-gray         0          0
     brown         0          0
      dark         0          0
     hazel         0          0
   unknown         0          0
    yellow         0          0

$grey
 eye_color feminine masculine
      blue         0          0
 blue-gray         0          0
     brown         0          0
      dark         0          0
     hazel         0          0
   unknown         0          0
    yellow         0          1

$none
 eye_color feminine masculine
      blue         0          1
 blue-gray         0          0
     brown         0          1
      dark         0          0
     hazel         0          0
   unknown         1          0
    yellow         0          1

$white
 eye_color feminine masculine
      blue         1          0
 blue-gray         0          0
     brown         0          1
      dark         0          0
     hazel         0          0
   unknown         0          0
    yellow         0          0
```

Für komplexere Analysen ist dies jedoch oft weniger praktisch als gruppierte Auswertungen mit `group_by()`.

Die adorn_*() Familie

Die wahre Stärke von `tabyl()` zeigt sich in Kombination mit den `adorn_*()`-Funktionen. Diese "dekorieren" die Tabelle mit zusätzlichen Informationen und Formatierungen.

adorn_totals() – Summenzeilen und -spalten

```
humans %>%
  tabyl(eye_color) %>%
  adorn_totals("row")
```

```
 eye_color  n    percent
      blue 12 0.34285714
 blue-gray  1 0.02857143
     brown 16 0.45714286
      dark  1 0.02857143
     hazel  2 0.05714286
   unknown  1 0.02857143
    yellow  2 0.05714286
     Total 35 1.00000000
```

Mit dem `name`-Argument können wir den Namen der Summenzeile anpassen:

```
humans %>%
  tabyl(eye_color) %>%
  adorn_totals("row", name = "Gesamt")
```

```
 eye_color  n    percent
      blue 12 0.34285714
 blue-gray  1 0.02857143
     brown 16 0.45714286
      dark  1 0.02857143
     hazel  2 0.05714286
   unknown  1 0.02857143
    yellow  2 0.05714286
    Gesamt 35 1.00000000
```

Bei Kreuztabellen können wir Zeilen, Spalten oder beides hinzufügen:

```
humans %>%
  tabyl(eye_color, gender) %>%
  adorn_totals(c("row", "col"))
```

```
 eye_color feminine masculine Total
      blue        3         9    12
 blue-gray        0         1     1
     brown        4        12    16
      dark        0         1     1
     hazel        1         1     2
   unknown        1         0     1
    yellow        0         2     2
     Total        9        26    35
```

adorn_percentages() – Prozente berechnen

Diese Funktion ersetzt die absoluten Zahlen durch Prozentanteile:

```
humans %>%
  tabyl(eye_color, gender) %>%
  adorn_percentages("row")  # Zeilenprozente
```

```
 eye_color feminine masculine
      blue     0.25       0.75
 blue-gray     0.00       1.00
     brown     0.25       0.75
      dark     0.00       1.00
     hazel     0.50       0.50
   unknown     1.00       0.00
    yellow     0.00       1.00
```

Das `denominator`-Argument bestimmt, worauf sich die Prozente beziehen:

- `"row"`: Zeilenprozente (jede Zeile summiert sich zu 100%)
- `"col"`: Spaltenprozente (jede Spalte summiert sich zu 100%)
- `"all"`: Gesamtprozente (die gesamte Tabelle summiert sich zu 100%)

```
humans %>%
  tabyl(eye_color, gender) %>%
  adorn_percentages("col")  # Spaltenprozente
```

```
 eye_color  feminine  masculine
      blue 0.3333333 0.34615385
 blue-gray 0.0000000 0.03846154
     brown 0.4444444 0.46153846
      dark 0.0000000 0.03846154
     hazel 0.1111111 0.03846154
   unknown 0.1111111 0.00000000
    yellow 0.0000000 0.07692308
```

adorn_pct_formatting() – Prozente formatieren

Nach `adorn_percentages()` sind die Werte noch Dezimalzahlen. Mit `adorn_pct_formatting()` werden sie schön formatiert:

```
humans %>%
  tabyl(eye_color, gender) %>%
  adorn_percentages("row") %>%
  adorn_pct_formatting(digits = 1)
```

```
 eye_color feminine masculine
      blue    25.0%     75.0%
 blue-gray     0.0%    100.0%
     brown    25.0%     75.0%
      dark     0.0%    100.0%
     hazel    50.0%     50.0%
   unknown   100.0%      0.0%
    yellow     0.0%    100.0%
```

Das `affix_sign`-Argument steuert, ob das Prozentzeichen angehängt wird:

```
humans %>%
  tabyl(eye_color, gender) %>%
  adorn_percentages("row") %>%
  adorn_pct_formatting(digits = 1, affix_sign = FALSE)
```

```
 eye_color feminine masculine
      blue     25.0       75.0
 blue-gray      0.0      100.0
     brown     25.0       75.0
      dark      0.0      100.0
     hazel     50.0       50.0
```

```
   unknown    100.0       0.0
    yellow      0.0     100.0
```

adorn_ns() – Fallzahlen zu Prozenten hinzufügen

Oft möchte man sowohl Prozente als auch absolute Zahlen sehen. `adorn_ns()` fügt die Fallzahlen in Klammern hinzu:

```
humans %>%
  tabyl(eye_color, gender) %>%
  adorn_percentages("row") %>%
  adorn_pct_formatting(digits = 0) %>%
  adorn_ns(position = "front")  # n vor Prozent
```

```
 eye_color feminine masculine
      blue 3  (25%)  9  (75%)
 blue-gray 0   (0%)  1 (100%)
     brown 4  (25%) 12  (75%)
      dark 0   (0%)  1 (100%)
     hazel 1  (50%)  1  (50%)
   unknown 1 (100%)  0   (0%)
    yellow 0   (0%)  2 (100%)
```

Mit `position = "rear"` erscheinen die Fallzahlen nach den Prozenten:

```
humans %>%
  tabyl(eye_color, gender) %>%
  adorn_percentages("row") %>%
  adorn_pct_formatting(digits = 0) %>%
  adorn_ns(position = "rear")  # n nach Prozent
```

```
 eye_color feminine masculine
      blue  25% (3)  75%  (9)
 blue-gray   0% (0) 100%  (1)
     brown  25% (4)  75% (12)
      dark   0% (0) 100%  (1)
     hazel  50% (1)  50%  (1)
   unknown 100% (1)   0%  (0)
    yellow   0% (0) 100%  (2)
```

adorn_title() – Tabellentitel hinzufügen

Für eine vollständige Beschriftung können wir Titel für Zeilen und Spalten hinzufügen:

```
humans %>%
  tabyl(eye_color, gender) %>%
  adorn_title(
    row_name = "Augenfarbe",
    col_name = "Geschlecht"
  )
```

```
            Geschlecht
 Augenfarbe   feminine masculine
       blue          3         9
  blue-gray          0         1
      brown          4        12
       dark          0         1
      hazel          1         1
    unknown          1         0
     yellow          0         2
```

Kombinierte Pipelines

Die `adorn_*()`-Funktionen lassen sich beliebig kombinieren. Eine typische Pipeline sieht so aus:

```
humans %>%
  tabyl(eye_color, gender) %>%
  adorn_totals(c("row", "col")) %>%
  adorn_percentages("row") %>%
  adorn_pct_formatting(digits = 1) %>%
  adorn_ns() %>%
  adorn_title(row_name = "Augenfarbe", col_name = "Geschlecht")
```

```
            Geschlecht
 Augenfarbe   feminine   masculine       Total
       blue  25.0% (3)  75.0%  (9) 100.0% (12)
  blue-gray   0.0% (0) 100.0%  (1) 100.0%  (1)
      brown  25.0% (4)  75.0% (12) 100.0% (16)
       dark   0.0% (0) 100.0%  (1) 100.0%  (1)
      hazel  50.0% (1)  50.0%  (1) 100.0%  (2)
    unknown 100.0% (1)   0.0%  (0) 100.0%  (1)
     yellow   0.0% (0) 100.0%  (2) 100.0%  (2)
      Total  25.7% (9)  74.3% (26) 100.0% (35)
```

Übung: Kreuztabellen und adorn_*()

Arbeite mit dem `humans`-Datensatz:

a) Erstelle eine Kreuztabelle von `gender` (Zeilen) und `eye_color` (Spalten) mit einer Summenzeile.

b) Erweitere die Tabelle aus a) um Spaltenprozente (jede Spalte = 100%), formatiert mit einer Dezimalstelle.

c) Füge zusätzlich die absoluten Fallzahlen hinzu (Position: nach den Prozenten).

Lösungsvorschlag

```
# a) Kreuztabelle mit Summenzeile
humans %>%
  tabyl(gender, eye_color) %>%
  adorn_totals("row")
```

```
    gender blue blue-gray brown dark hazel unknown yellow
  feminine    3         0     4    0     1       1      0
 masculine    9         1    12    1     1       0      2
     Total   12         1    16    1     2       1      2
```

```
# b) Mit Spaltenprozenten
humans %>%
  tabyl(gender, eye_color) %>%
  adorn_totals("row") %>%
  adorn_percentages("col") %>%
  adorn_pct_formatting(digits = 1)
```

```
    gender   blue blue-gray  brown   dark  hazel unknown yellow
  feminine  25.0%      0.0%  25.0%   0.0%  50.0%  100.0%   0.0%
 masculine  75.0%    100.0%  75.0% 100.0%  50.0%    0.0% 100.0%
     Total 100.0%    100.0% 100.0% 100.0% 100.0%  100.0% 100.0%
```

```
# c) Mit Fallzahlen
humans %>%
  tabyl(gender, eye_color) %>%
  adorn_totals("row") %>%
  adorn_percentages("col") %>%
  adorn_pct_formatting(digits = 1) %>%
  adorn_ns(position = "rear")
```

```
    gender        blue  blue-gray       brown        dark      hazel    unknown
  feminine  25.0%  (3)   0.0% (0)  25.0%  (4)   0.0% (0)  50.0% (1) 100.0% (1)
 masculine  75.0%  (9) 100.0% (1)  75.0% (12) 100.0% (1)  50.0% (1)   0.0% (0)
     Total 100.0% (12) 100.0% (1) 100.0% (16) 100.0% (1) 100.0% (2) 100.0% (1)
     yellow
   0.0% (0)
 100.0% (2)
 100.0% (2)
```

Fortgeschritten: Praxistipps

Named Vectors für Recoding

Wenn Variablen kryptische Codes haben (z.B. `var1`, `var2`, …), möchten wir sie oft mit verständlichen Labels versehen. Statt eines langen `case_when()` empfiehlt sich ein Named Vector:

```
# Named Vector definieren (wiederverwendbar!)
eye_labels <- c(
  "blue" = "Blau",
  "brown" = "Braun",
  "hazel" = "Haselnuss",
  "dark" = "Dunkel",
  "blue-gray" = "Blau-Grau"
)

# Anwendung
humans %>%
  mutate(eye_color_de = eye_labels[eye_color]) %>%
  tabyl(eye_color_de, show_na = FALSE)
```

```
 eye_color_de  n percent
         Blau 12 0.37500
    Blau-Grau  1 0.03125
        Braun 16 0.50000
       Dunkel  1 0.03125
    Haselnuss  2 0.06250
```

Dieser Ansatz ist:

- **Wiederverwendbar**: Der Vektor kann in mehreren Auswertungen genutzt werden
- **Zentral pflegbar**: Änderungen an einem Ort wirken sich überall aus
- **Übersichtlich**: Keine langen `case_when()`-Blöcke im Code

> Tipp: Labels in separater Datei
>
> Bei vielen Variablen lohnt es sich, alle Label-Vektoren in einer separaten R-Datei zu speichern (z.B. `labels.R`) und diese am Anfang des Skripts zu laden:
>
> ```
> source("labels.R")
> ```

Warnung: Mittelwert von Mittelwerten

Wenn man `adorn_totals()` auf Tabellen anwendet, die bereits aggregierte Werte enthalten, ist Vorsicht geboten. Das betrifft besonders **Mittelwerte**:

```
# Beispiel: Durchschnittliche Körpergröße nach Geschlecht
height_by_gender <- humans %>%
  group_by(gender) %>%
  summarise(
    n = n(),
    mean_height = mean(height, na.rm = TRUE)
  )

height_by_gender
```

```
# A tibble: 2 × 3
  gender        n mean_height
  <chr>     <int>       <dbl>
1 feminine      9        164.
2 masculine    26        182.
```

```
# FALSCH: adorn_totals() summiert auch den Mittelwert!
height_by_gender %>%
  adorn_totals("row")
```

```
    gender  n mean_height
  feminine  9    163.5714
 masculine 26    182.3913
     Total 35    345.9627
```

Das Problem: `adorn_totals()` addiert einfach die Zeilen. Bei der Spalte `n` ist das korrekt, aber bei `mean_height` ergibt die Summe keinen Sinn!

> ! Der Mittelwert von Mittelwerten ist nicht der Gesamtmittelwert!
>
> Wenn die Gruppen unterschiedlich groß sind, führt der einfache Durchschnitt der Gruppenmittelwerte zu einer **Verzerrung**. Der korrekte Gesamtmittelwert muss gewichtet berechnet werden.

Hier ein Beispiel zur Veranschaulichung:

```
# Gruppe A: 100 Personen, Durchschnitt 20
# Gruppe B: 10 Personen, Durchschnitt 30

# Falscher "Gesamtmittelwert": (20 + 30) / 2 = 25

# Korrekter Gesamtmittelwert:
# (100 * 20 + 10 * 30) / (100 + 10) = 2300 / 110 ≈ 20.9

tibble(
  Gruppe = c("A", "B"),
  n = c(100, 10),
  Mittelwert = c(20, 30)
) %>%
  adorn_totals("row")  # Zeigt 25 statt 20.9!
```

```
 Gruppe   n Mittelwert
      A 100         20
      B  10         30
  Total 110         50
```

Lösung: Die Gesamtzeile bei Mittelwerten separat und korrekt berechnen:

```
# Schritt 1: Gruppierte Mittelwerte
height_by_gender <- humans %>%
  group_by(gender) %>%
  summarise(
    n = n(),
    mean_height = mean(height, na.rm = TRUE)
  )

# Schritt 2: Gesamtzeile separat berechnen
gesamt <- humans %>%
  summarise(
    gender = "Total",
    n = n(),
```

```
    mean_height = mean(height, na.rm = TRUE)
  )

# Schritt 3: Zusammenfügen
bind_rows(height_by_gender, gesamt)
```

```
# A tibble: 3 × 3
  gender        n mean_height
  <chr>     <int>       <dbl>
1 feminine      9        164.
2 masculine    26        182.
3 Total        35        178
```

Übung: Praxisanwendung

Verwende den vollständigen `starwars`-Datensatz (nicht nur Menschen):

a) Erstelle eine Häufigkeitstabelle für `species`, aber zeige nur die 5 häufigsten Spezies. Alle anderen sollen unter “Andere” zusammengefasst werden. Tipp: Nutze `fct_lump_n()` aus dem {forcats}-Paket.

b) Füge eine Summenzeile mit dem Namen “Gesamt” hinzu und formatiere die Prozente mit einer Dezimalstelle.

Lösungsvorschlag

```
# a) + b) Häufigkeitstabelle der Top-5 Spezies
starwars %>%
  mutate(species = fct_lump_n(species, n = 5, other_level = "Andere")) %>%
  tabyl(species, show_na = FALSE) %>%
  adorn_totals("row", name = "Gesamt") %>%
  adorn_pct_formatting(digits = 1)
```

```
  species  n percent
    Droid  6    7.2%
   Gungan  3    3.6%
    Human 35   42.2%
 Kaminoan  2    2.4%
 Mirialan  2    2.4%
  Twi'lek  2    2.4%
  Wookiee  2    2.4%
   Zabrak  2    2.4%
   Andere 29   34.9%
   Gesamt 83  100.0%
```

Zusammenfassung

In diesem Kapitel haben wir drei Wege kennengelernt, Häufigkeitstabellen in R zu erstellen, und gesehen, warum `janitor::tabyl()` in den meisten Fällen die beste Wahl ist.

ℹ Wichtige Erkenntnisse

Vergleich der Methoden:

Aspekt	`table()`	`count()`	`tabyl()`
Rückgabetyp	table-Objekt	tibble	tibble
Prozente	Nein	Manuell	Automatisch
NA-Handling	Eingeschränkt	Manuell	`show_na`
Summenzeile	Manuell	Manuell	`adorn_totals()`
Kreuztabellen	Ja	Umständlich	Ja
Weiterverarbeitung	Umständlich	Gut	Sehr gut

Die wichtigsten `tabyl()`-Features:

- `tabyl(df, var)`: Einweg-Tabelle mit n, percent, valid_percent
- `tabyl(df, var1, var2)`: Kreuztabelle
- `show_na = FALSE`: NA-Werte ausblenden
- `show_missing_levels = TRUE`: Leere Faktor-Levels anzeigen

Die `adorn_*()` Familie:

- `adorn_totals()`: Summenzeile/-spalte hinzufügen
- `adorn_percentages()`: Prozente berechnen (row/col/all)
- `adorn_pct_formatting()`: Prozente formatieren
- `adorn_ns()`: Fallzahlen zu Prozenten hinzufügen
- `adorn_title()`: Zeilen-/Spaltentitel setzen

Praxistipps:

- Named Vectors für Recoding statt langes `case_when()`
- Vorsicht bei `adorn_totals()` und Mittelwerten – der Mittelwert von Mittelwerten ist nicht der Gesamtmittelwert!
- Die typische Pipeline:

```
tabyl() %>% adorn_totals() %>% adorn_percentages() %>% adorn_pct_formatting() %>% adorn_ns()
```

Weiterführende Ressourcen:

- janitor Package Dokumentation
- tabyl Vignette

Bibliography