3. Frequency Tables

Counting and Tabulating with janitor::tabyl()
Dr. Paul Schmidt

To install and load all packages used in this chapter, run the following code:

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

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

Introduction

Frequency tables are one of the most fundamental tools in data analysis. How often does each category occur? How are values distributed across different groups? We constantly ask such questions – whether in quality control, survey analysis, or simply to get an initial overview of the data.

R offers several ways to create frequency tables. In this chapter, we start with the basics (`table()` and `count()`) to then understand why `janitor::tabyl()` is the more elegant and practical solution in most cases.

Example Data

For this chapter, we use the `starwars` dataset from the {dplyr} package. It contains information about 87 characters from the Star Wars universe:

```r
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",…
```

The dataset has both categorical variables (like `species`, `sex`, `homeworld`) and numerical variables (like `height`, `mass`). For most examples, we filter to humans (`species == "Human"`) to keep the outputs manageable:

```
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>
```

The Classic Way: table()

The `table()` function is built into base R and creates simple frequency tables:

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

```

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

This works, but has several disadvantages:

1. **Not a data.frame**: The result is a `table` object, not a tibble/data.frame. It cannot be directly processed with tidyverse functions.
2. **No percentages**: We only get absolute counts, no relative frequencies.
3. **Awkward syntax**: With multiple variables, it quickly becomes unwieldy.

You can convert the result to a data.frame, but it's cumbersome:

```r
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
```

The column names are not intuitive (`Var1`, `Freq`), and we have to calculate percentages ourselves.

The Tidyverse Way: count() + mutate()

With `dplyr::count()`, we get a tibble directly:

```r
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
```

That's already better! If we want percentages, we add them with `mutate()`:

```r
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%
```

And if we want a total row, we have to calculate it separately and then append it:

```
# Step 1: Calculate frequencies per category
per_eye_color <- humans %>%
  count(eye_color) %>%
  mutate(percent = n / sum(n))

per_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
```

```
# Step 2: Create total row separately
total <- tibble(
  eye_color = "Total",
  n = sum(per_eye_color$n),
  percent = 1
)

total
```

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

```
# Step 3: Combine
bind_rows(per_eye_color, total)
```

```
# 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
```

This works, but it's a lot of typing for such a common task. This is where `tabyl()` comes in.

janitor::tabyl() – The Elegant Solution

The `tabyl()` function from the {janitor} package was designed exactly for this use case. It combines the best features of `table()` and `count()` and adds additional useful features.

One-Way Table (Single 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
```

With a single function call, we get:

- **n**: The absolute frequency
- **percent**: The relative proportion (as a decimal)

The result is a tibble that we can process directly.

Controlling NA Values

Let's look at a variable with missing values – `homeworld` has several NA entries:

```
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
```

By default, `tabyl()` shows NA values as a separate category. Note the two percent columns:

- **percent**: Proportion based on all rows (including NA)
- **valid_percent**: Proportion based on valid values (excluding NA)

With `show_na = FALSE`, we can hide NA values:

```
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
```

When we set `show_na = FALSE`, there's only one percent column since both values would be identical.

Showing Empty Categories

If a variable is defined as a factor, there may be levels that don't appear in the dataset. With `show_missing_levels = TRUE`, these are still displayed:

```
# Example: Factor with a level that doesn't occur
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
```

The level "green" doesn't occur in humans, but is still shown with n=0. This is particularly useful for survey data where certain response categories may not have been chosen by anyone, but should still appear in the report.

Exercise: One-Way Tables

Create the following tables using the `humans` dataset:

a) A frequency table for the variable `gender`.

b) A frequency table for `homeworld` with NA values hidden.

Solution

```
# a) Gender distribution
humans %>%
 tabyl(gender)
```

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

```
# b) Homeworlds without 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
```

Two-Way Tables (Cross-Tabulations)

With two variables, `tabyl()` automatically creates a cross-tabulation:

```
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
```

The first variable (`eye_color`) defines the rows, the second (`gender`) the columns. The result shows the absolute frequencies for each combination.

Three-Way Tables

With three variables, `tabyl()` creates a list of cross-tabulations – one for each level of the third 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
```

For more complex analyses, however, this is often less practical than grouped analyses with `group_by()`.

The adorn_*() Family

The true power of `tabyl()` shows itself in combination with the `adorn_*()` functions. These "adorn" the table with additional information and formatting.

adorn_totals() – Total Rows and Columns

```r
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
```

With the `name` argument, we can customize the name of the total row:

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

```
 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
```

For cross-tabulations, we can add rows, columns, or both:

```r
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() – Calculate Percentages

This function replaces absolute counts with percentage proportions:

```r
humans %>%
  tabyl(eye_color, gender) %>%
  adorn_percentages("row")  # Row percentages
```

```
 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
```

The `denominator` argument determines what the percentages are based on:

- `"row"`: Row percentages (each row sums to 100%)
- `"col"`: Column percentages (each column sums to 100%)
- `"all"`: Overall percentages (the entire table sums to 100%)

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

```
 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() – Format Percentages

After `adorn_percentages()`, the values are still decimals. With `adorn_pct_formatting()`, they are nicely formatted:

```
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%
```

The `affix_sign` argument controls whether the percent sign is appended:

```
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() – Add Case Counts to Percentages

Often we want to see both percentages and absolute numbers. `adorn_ns()` adds the case counts in parentheses:

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

```
 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%)
```

With `position = "rear"`, the case counts appear after the percentages:

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

```
 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() – Add Table Titles

For complete labeling, we can add titles for rows and columns:

```
humans %>%
  tabyl(eye_color, gender) %>%
  adorn_title(
    row_name = "Eye Color",
    col_name = "Gender"
  )
```

```
             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
```

Combined Pipelines

The `adorn_*()` functions can be combined as needed. A typical pipeline looks like this:

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

```
              Gender                        
 Eye Color  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)
```

Exercise: Cross-Tabulations and adorn_*()

Work with the `humans` dataset:

a) Create a cross-tabulation of `gender` (rows) and `eye_color` (columns) with a total row.

b) Extend the table from a) with column percentages (each column = 100%), formatted with one decimal place.

c) Additionally add the absolute case counts (position: after the percentages).

Solution

```
# a) Cross-tabulation with total row
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) With column percentages
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) With case counts
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)
```

Advanced: Practical Tips

Named Vectors for Recoding

When variables have cryptic codes (e.g., `var1`, `var2`, …), we often want to label them with understandable names. Instead of a long `case_when()`, a named vector is recommended:

```
# Define named vector (reusable!)
eye_labels <- c(
  "blue" = "Blue",
  "brown" = "Brown",
  "hazel" = "Hazel",
  "dark" = "Dark",
  "blue-gray" = "Blue-Gray"
)

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

```
 eye_color_label  n percent
            Blue 12 0.37500
       Blue-Gray  1 0.03125
           Brown 16 0.50000
            Dark  1 0.03125
           Hazel  2 0.06250
```

This approach is:

- **Reusable**: The vector can be used in multiple analyses
- **Centrally maintainable**: Changes in one place affect everywhere
- **Clear**: No long `case_when()` blocks in the code

Tip: Labels in Separate File

With many variables, it's worthwhile to store all label vectors in a separate R file (e.g., `labels.R`) and load it at the beginning of the script:

```
source("labels.R")
```

Warning: Mean of Means

When applying `adorn_totals()` to tables that already contain aggregated values, caution is required. This particularly applies to **means**:

```
# Example: Average height by gender
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.
```

```
# WRONG: adorn_totals() also sums the mean!
height_by_gender %>%
  adorn_totals("row")
```

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

The problem: `adorn_totals()` simply adds the rows. For the `n` column, this is correct, but for `mean_height`, the sum makes no sense!

! The Mean of Means Is Not the Overall Mean!

When groups have different sizes, the simple average of group means leads to **bias**. The correct overall mean must be calculated as a weighted average.

Here's an example for illustration:

```
# Group A: 100 people, average 20
# Group B: 10 people, average 30

# Wrong "overall mean": (20 + 30) / 2 = 25

# Correct overall mean:
# (100 * 20 + 10 * 30) / (100 + 10) = 2300 / 110 ≈ 20.9

tibble(
  Group = c("A", "B"),
  n = c(100, 10),
  Mean = c(20, 30)
) %>%
  adorn_totals("row")  # Shows 25 instead of 20.9!
```

```
 Group   n Mean
     A 100   20
     B  10   30
 Total 110   50
```

Solution: Calculate the total row for means separately and correctly:

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

# Step 2: Calculate total row separately
total <- humans %>%
  summarise(
    gender = "Total",
    n = n(),
    mean_height = mean(height, na.rm = TRUE)
  )
```

```
# Step 3: Combine
bind_rows(height_by_gender, total)
```

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

Exercise: Practical Application

Use the complete `starwars` dataset (not just humans):

a) Create a frequency table for `species`, but show only the 5 most common species. All others should be combined under “Other”. Tip: Use `fct_lump_n()` from the {forcats} package.

b) Add a total row named “Total” and format the percentages with one decimal place.

Solution

```
# a) + b) Frequency table of top 5 species
starwars %>%
  mutate(species = fct_lump_n(species, n = 5, other_level = "Other")) %>%
  tabyl(species, show_na = FALSE) %>%
  adorn_totals("row", name = "Total") %>%
  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%
    Other 29   34.9%
    Total 83  100.0%
```

Summary

In this chapter, we learned three ways to create frequency tables in R and saw why `janitor::tabyl()` is the best choice in most cases.

i Key Takeaways

Comparison of Methods:

Aspect	`table()`	`count()`	`tabyl()`
Return type	table object	tibble	tibble
Percentages	No	Manual	Automatic
NA handling	Limited	Manual	`show_na`
Total row	Manual	Manual	`adorn_totals()`
Cross-tabulations	Yes	Awkward	Yes
Further processing	Awkward	Good	Very good

Key `tabyl()` Features:

- `tabyl(df, var)`: One-way table with n, percent, valid_percent
- `tabyl(df, var1, var2)`: Cross-tabulation
- `show_na = FALSE`: Hide NA values
- `show_missing_levels = TRUE`: Show empty factor levels

The `adorn_*()` Family:

- `adorn_totals()`: Add total row/column
- `adorn_percentages()`: Calculate percentages (row/col/all)
- `adorn_pct_formatting()`: Format percentages
- `adorn_ns()`: Add case counts to percentages
- `adorn_title()`: Set row/column titles

Practical Tips:

- Named vectors for recoding instead of long `case_when()`
- Caution with `adorn_totals()` and means – the mean of means is not the overall mean!
- Typical pipeline:

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

Further Resources:

- janitor Package Documentation
- tabyl Vignette

Bibliography