Chapter 5 & 6

5.2.1 Exercises

1. For each of the sample tables, describe what each observation and each column represents.

- Table 1 observation - Each row represents a unique combination of country and year.
- Table 2 observation - Each row represents a specific attribute (cases or population) for a country and year.
- Table 3 observation - Each row represents a unique combination of country and year.

2. Sketch out the process you'd use to calculate the rate for table2 and table3. You will need to perform four operations:
    1. Extract the number of TB cases per country per year.
    2. Extract the matching population per country per year.
    3. Divide cases by population, and multiply by 10000.
    4. Store back in the appropriate place.

- **For table2 (Long Format)**

3. **Extract cases**
    1. Filter rows where type == "cases" and store them separately.
4. **Extract population**
    1. Filter rows where type == "population" and store them separately.
5. **Join cases and population**
    1. Merge both tables on country and year to match cases with population.
6. **Compute the rate**
    1. Use the formula:rate=(casespopulation)×10,000rate=(populationcases)×10,000
7. **Store in a new table**
    1. Create a new column rate and store the results.

- **Extract cases and population**
    - The rate column in table3 contains values as strings like "cases/population".
    - Split the string on / to separate cases and population.
- **Convert to numeric**
    - Ensure cases and population are stored as integers.
- **Compute the rate**
    - Apply the same formula:rate=(casespopulation)×10,000rate=(populationcases)×10,000

- **Store in a new column**
    - Replace the original rate column with the computed numerical rate.

Executables

```
# Chapter 5  Data tidying

#5.1.1 Prerequisites ......................................

library(tidyverse)

#5.2 Tidy data .........................................

table1

table2

table3

table1 |>
  mutate(rate = cases / population * 10000)

table1 |>
  group_by(year) |>
  summarize(total_cases = sum(cases))

ggplot(table1, aes(x = year, y = cases)) +
  geom_line(aes(group = country), color = "grey50") +
  geom_point(aes(color = country, shape = country)) +
  scale_x_continuous(breaks = c(1999, 2000))

#5.3.1 Data in column names .............................................
billboard

billboard |>
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    values_to = "rank"
  )

billboard |>
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    values_to = "rank",
    values_drop_na = TRUE
  )

billboard_longer <- billboard |>
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    values_to = "rank",
    values_drop_na = TRUE
  ) |>
  mutate(
    week = parse_number(week)
  )
billboard_longer

billboard_longer |>
  ggplot(aes(x = week, y = rank, group = track)) +
  geom_line(alpha = 0.25) +
  scale_y_reverse()

#5.3.2 How does pivoting work? ..........................................

df <- tribble(
  ~id,  ~bp1, ~bp2,
  "A",  100,  120,
  "B",  140,  115,
  "C",  120,  125
)

df |>
```

```
  pivot_longer(
    cols = bp1:bp2,
    names_to = "measurement",
    values_to = "value"
  )

#5.3.3 Many variables in column names .....................................

who2

who2 |>
  pivot_longer(
    cols = !(country:year),
    names_to = c("diagnosis", "gender", "age"),
    names_sep = "_",
    values_to = "count"
  )

#5.3.4 Data and variable names in the column headers ......................

household

household |>
  pivot_longer(
    cols = !family,
    names_to = c(".value", "child"),
    names_sep = "_",
    values_drop_na = TRUE
  )

#5.4 Widening data ................................

cms_patient_experience

cms_patient_experience |>
  distinct(measure_cd, measure_title)

cms_patient_experience |>
  pivot_wider(
    names_from = measure_cd,
    values_from = prf_rate
  )

cms_patient_experience |>
  pivot_wider(
    id_cols = starts_with("org"),
    names_from = measure_cd,
    values_from = prf_rate
  )

#5.4.1 How does pivot_wider() work? ...........................

df <- tribble(
  ~id, ~measurement, ~value,
  "A",        "bp1",    100,
  "B",        "bp1",    140,
  "B",        "bp2",    115,
  "A",        "bp2",    120,
  "A",        "bp3",    105
)

df |>
  pivot_wider(
    names_from = measurement,
    values_from = value
  )

df |>
  distinct(measurement) |>
  pull()

df |>
  select(-measurement, -value) |>
  distinct()

df |>
  select(-measurement, -value) |>
  distinct() |>
  mutate(x = NA, y = NA, z = NA)
```

```
df <- tribble(
  ~id, ~measurement, ~value,
  "A",        "bp1",    100,
  "A",        "bp1",    102,
  "A",        "bp2",    120,
  "B",        "bp1",    140,
  "B",        "bp2",    115
)

df |>
  pivot_wider(
    names_from = measurement,
    values_from = value
  )

df |>
  group_by(id, measurement) |>
  summarize(n = n(), .groups = "drop") |>
  filter(n > 1)

#Chapter 6  Workflow: scripts and projects ............................

#6.1.1 Running code

library(dplyr)
library(nycflights13)

not_cancelled <- flights |>
  filter(!is.na(dep_delay), !is.na(arr_delay))

not_cancelled |>
  group_by(year, month, day) |>
  summarize(mean = mean(dep_delay))

#6.1.3 Saving and naming ....................

# File names should be machine readable: avoid spaces, symbols, and special characters. Don’t rely on case sensitivity to distinguish files.
# File names should be human readable: use file names to describe what’s in the file.
```

```r
# File names should play well with default ordering: start file names with numbers so that alphabetical sorting puts them in the order they get used.

#Here’s a better way of naming and organizing the same set of files:

#   01-load-data.R
# 02-exploratory-analysis.R
# 03-model-approach-1.R
# 04-model-approach-2.R
# fig-01.png
# fig-02.png
# report-2022-03-20.qmd
# report-2022-04-02.qmd
# report-draft-notes.txt

#6.2 Projects .............................................

# To handle these real life situations, you need to make two decisions:
#
#   What is the source of truth? What will you save as your lasting record of what happened?
#
#   Where does your analysis live?

#6.2.2 Where does your analysis live? ...................

getwd()

#6.2.4 Relative and absolute paths .........................

# A relative path is relative to the working directory and should be the only type

#never use absolute paths in your scripts, because they hinder sharing
# no one else will have exactly the same directory configuration as you.
```