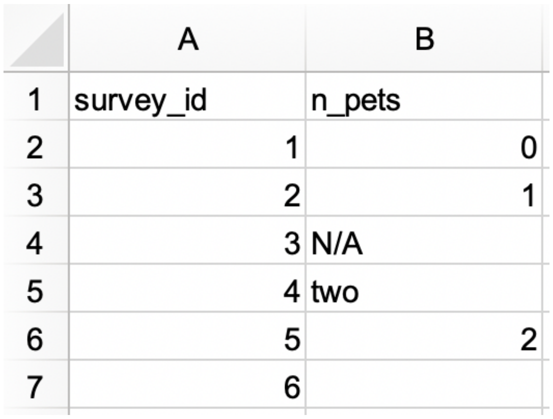
Chapter 20 & 21: Exercises & Executables

20.2.9 Exercises

1. In an Excel file, create the following dataset and save it as survey.xlsx. Alternatively, you can download it as an Excel file from here.



	A	B
1	survey_id	n_pets
2	1	0
3	2	1
4	3	N/A
5	4	two
6	5	2
7	6	

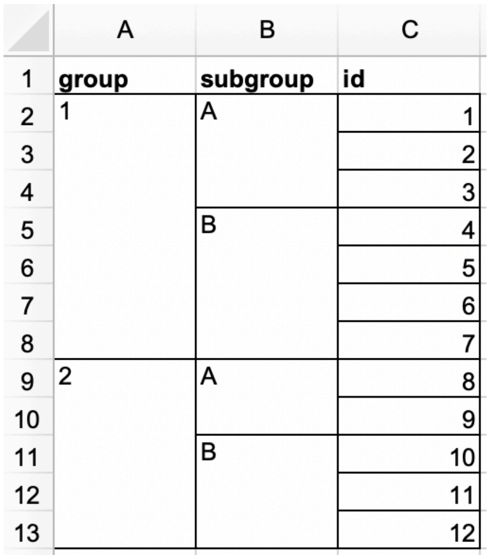
Then, read it into R, with survey_id as a character variable and n_pets as a numerical variable.

```
#> # A tibble: 6 × 2

#>   survey_id n_pets

#>   <chr>      <dbl>

#> 1 1            0

#> 2 2            1

#> 3 3           NA

#> 4 4            2

#> 5 5            2
        #> 6 6           NA
```

- Create the Excel file survey.xlsx with survey_id and n_pets.
- Use read_excel() from the readxl package to read it into R.
- Convert survey_id to character and clean n_pets using parse_number() to handle "N/A" and "two":

```
-   library(readxl)
-   library(tidyverse)
-
-   survey <- read_excel("survey.xlsx") %>%
-     mutate(
-       survey_id = as.character(survey_id),
-       n_pets = readr::parse_number(n_pets)
  )
```

2. In another Excel file, create the following dataset and save it as roster.xlsx. Alternatively, you can download it as an Excel file from here.



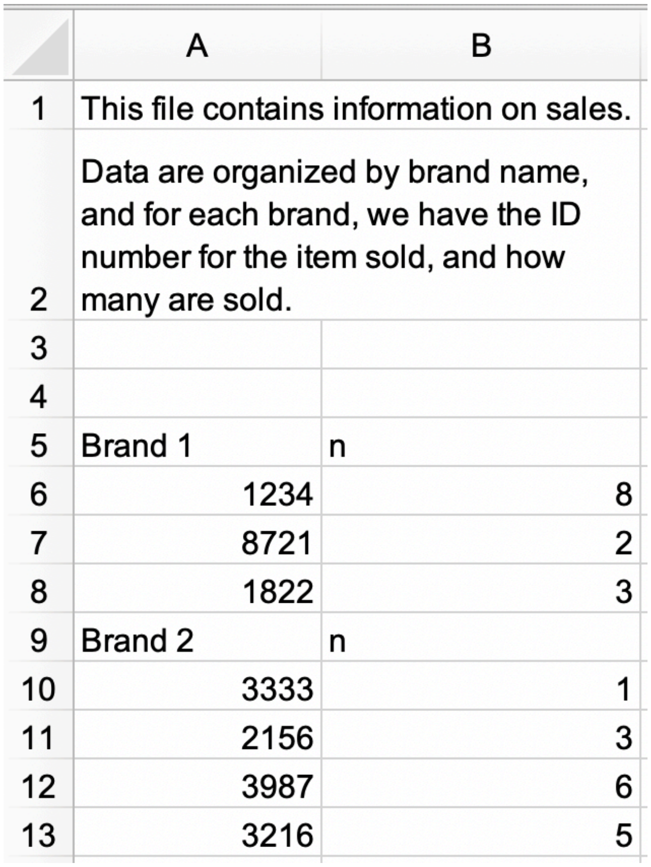
	A	B	C
1	group	subgroup	id
2	1	A	1
3			2
4			3
5		B	4
6			5
7			6
8			7
9	2	A	8
10			9
11		B	10
12			11
13			12

Then, read it into R. The resulting data frame should be called roster and should look like the following.

```
#> # A tibble: 12 × 3

#>   group subgroup    id
#>    <dbl> <chr>    <dbl>
#>  1     1 A            1
#>  2     1 A            2
#>  3     1 A            3
#>  4     1 B            4
#>  5     1 B            5
#>  6     1 B            6
#>  7     1 B            7
#>  8     2 A            8
#>  9     2 A            9
#> 10     2 B           10
#> 11     2 B           11
#> 12     2 B           12
```

```
-   library(readxl)
-   library(tidyverse)
-
-   # Read the Excel file
-   roster_raw <- read_excel("roster.xlsx")
-
-   # Fill down missing group and subgroup values
-   roster <- roster_raw %>%
-     fill(group, subgroup)
```

3. In a new Excel file, create the following dataset and save it as sales.xlsx. Alternatively, you can download it as an Excel file from here.



	A	B
1	This file contains information on sales.	
2	Data are organized by brand name, and for each brand, we have the ID number for the item sold, and how many are sold.	
3		
4		
5	Brand 1	n
6	1234	8
7	8721	2
8	1822	3
9	Brand 2	n
10	3333	1
11	2156	3
12	3987	6
13	3216	5

a. Read sales.xlsx in and save as sales. The data frame should look like the following, with id and n as column names and with 9 rows.

```
#> # A tibble: 9 × 2

#>   id      n

#>   <chr>   <chr>

#> 1 Brand 1 n

#> 2 1234    8

#> 3 8721    2

#> 4 1822    3

#> 5 Brand 2 n
#> 6 3333    1

#> 7 2156    3

#> 8 3987    6
#> 9 3216    5
```

1. Read sales.xlsx and view the initial data

```
-    library(readxl)
-    sales <- read_excel("sales.xlsx", skip = 3)
```

2. Clean and reshape the data into tidy format

```
    -    library(tidyverse)

    -    sales_tidy <- sales %>%
    -      fill(id) %>%                 # fill down brand names
    -      filter(id != "n") %>%         # remove rows with "n"
    -      mutate(brand = id) %>%
    -      fill(brand) %>%               # fill down brand again
    -      filter(id != brand) %>%       # remove brand rows
    -      mutate(
    -        id = as.numeric(id),        # convert ID to number
    -        brand = str_extract(brand, "Brand \\d+")  # keep brand only
    -      ) %>%
    -      select(brand, id, n)
```

b. Modify sales further to get it into the following tidy format with three columns (brand, id, and n) and 7 rows of data. Note that id and n are numeric, brand is a character variable.

```
#> # A tibble: 7 × 3

#>   brand      id     n

#>  <chr>   <dbl> <dbl>

#> 1 Brand 1  1234     8
#> 2 Brand 1  8721     2

#> 3 Brand 1  1822     3

#> 4 Brand 2  3333     1

#> 5 Brand 2  2156     3

#> 6 Brand 2  3987     6
        #> 7 Brand 2  3216     5
```

```
-   library(tidyr)
-   library(dplyr)
-
-   sales_tidy <- sales %>%
-     pivot_longer(
-       cols = everything(),
-       names_to = c("brand", ".value"),
-       names_pattern = "(brand\\d+)_(id|n)"
-     )
```

4. Recreate the bake_sale data frame, write it out to an Excel file using the write.xlsx()function from the openxlsx package.

```
-   library(tibble)
-   library(openxlsx)
-
-   # Step 1: Create the bake_sale data frame
-   bake_sale <- tibble(
-     item = c("brownie", "cupcake", "cookie", "muffin"),
-     quantity = c(10, 5, 8, 6),
-     price = c(2.00, 2.50, 1.50, 2.00)
-   )
-
-   # Step 2: Write it to an Excel file
-   write.xlsx(bake_sale, "bake_sale.xlsx")
```

5. In Chapter 7 you learned about the janitor::clean_names() function to turn column names into snake case. Read the students.xlsx file that we introduced earlier in this section and use this function to “clean” the column names.

```
-   library(readxl)
-   library(janitor)
```

-
- students <- read_excel("students.xlsx")
-
- students_clean <- students %>%
-   clean_names()

6. What happens if you try to read in a file with .xlsx extension with read_xls()?|

- If you use read_xls() on a .xlsx file, it will give an error because:
- read_xls() is only meant for older .xls files.
- .xlsx is a newer format (Excel 2007 and later).
- You must use read_xlsx() to read .xlsx files.

20.3.6 Exercises

1. Read the students dataset from earlier in the chapter from Excel and also from Google Sheets, with no additional arguments supplied to the read_excel() and read_sheet()functions. Are the resulting data frames in R exactly the same? If not, how are they different?

- read_excel() guesses the types of columns based on the first 1000 rows.
- read_sheet() always reads all columns as text by default if types are ambiguous, unless you specify otherwise.

2. Read the Google Sheet titled survey from https://pos.it/r4ds-survey, with survey_id as a character variable and n_pets as a numerical variable.

- library(googlesheets4)
- library(tidyverse)
- survey <- read_sheet("https://pos.it/r4ds-survey")
- survey <- survey %>%
-   mutate(
-     survey_id = as.character(survey_id),
-     n_pets = readr::parse_number(n_pets) # converts 'two' and 'N/A' to NA
-   )

3. Read the Google Sheet titled roster from https://pos.it/r4ds-roster. The resulting data frame should be called roster and should look like the following.

```
#> # A tibble: 12 × 3
#>    group subgroup    id
#>    <dbl> <chr>    <dbl>
#>  1     1 A            1
#>  2     1 A            2
#>  3     1 A            3
#>  4     1 B            4
#>  5     1 B            5
#>  6     1 B            6
#>  7     1 B            7
#>  8     2 A            8
#>  9     2 A            9
#> 10     2 B           10
#> 11     2 B           11
#> 12     2 B           12
```

- We use read_sheet() to read the roster sheet.
- R automatically guesses the column types: group and id become numbers, subgroup becomes text.
- The missing cells (blank rows) are filled down automatically based on the structure of the sheet.
- The result is a tidy tibble with 12 rows and 3 columns — exactly matching the example shown.

21.5.10 Exercises

1. What is `distinct()` translated to? How about `head()`?

- distinct() in dbplyr is translated to SELECT DISTINCT in SQL.
- head() is translated to LIMIT in SQL.

2. Explain what each of the following SQL queries do and try to recreate them using dbplyr.

```
SELECT *
FROM flights
```

```
WHERE dep_delay < arr_delay

SELECT *, distance / (air_time / 60) AS speed

      FROM flights
```

1. First SQL Query:
    - SELECT *
    - FROM flights
    - WHERE dep_delay < arr_delay

Meaning:

- Select all columns from the flights table, but only rows where departure delay is less than arrival delay.

Recreate with dplyr:
- flights %>%
- filter(dep_delay < arr_delay)

2. Second SQL Query:
    - SELECT *, distance / (air_time / 60) AS speed
    - FROM flights

Meaning:

- Select all columns from flights, and create a new column called speed where speed = distance divided by air_time (in minutes).

Recreate with dplyr:

- flights %>%
- mutate(speed = distance / (air_time / 60))