Chapter 7 and 8

7.2.4 Exercises

1. What function would you use to read a file where fields were separated with “|”?
- To read a file where fields are separated by a pipe (|) character, you can use the read_delim() function from the readr package in R. This function allows you to specify the delimiter used in your data file.

2. Apart from file, skip, and comment, what other arguments do read_csv() and read_tsv() have in common?
- In addition to file, skip, and comment, the read_csv() and read_tsv() functions from the readr package share several other common arguments:
- **col_names**: Specifies whether the first row of the data contains column names. If set to FALSE, readr assigns default names (X1, X2, etc.) to the columns. Alternatively, you can provide a character vector to define custom column names.
- **col_types**: Allows explicit specification of the data types for each column, preventing readr from guessing them automatically.
- **na**: Defines which strings should be interpreted as missing values (NA). By default, readr recognizes empty strings as NA, but you can customize this by providing a character vector of strings to consider as missing.

3. What are the most important arguments to read_fwf()?
- The most important arguments for read_fwf() are:
- **file**: Path to the file.
- **col_positions**: Defines column positions using fwf_widths() or fwf_positions().
- **col_types**: Specifies data types for each column.
- **skip**: Number of lines to skip.
- **n_max**: Limits the number of lines read.

4. Sometimes strings in a CSV file contain commas. To prevent them from causing problems, they need to be surrounded by a quoting character, like " or '. By default, read_csv()assumes that the quoting character will be ". To read the following text into a data frame, what argument to read_csv() do you need to specify? - "x,y\n1,'a,b'"

- To correctly read the given text into a data frame, you need to specify the `quote` argument in `read_csv()` to match the quoting character used in the data. Since the text uses a single quote (`'`) instead of the default double quote (`"`), set `quote = "'"`:
- read_csv("x,y\n1,'a,b'", quote = "'")

5. Identify what is wrong with each of the following inline CSV files. What happens when you run the code?

1. `read_csv("a,b\n1,2,3\n4,5,6")`
- **Issue**: More values than columns.
- **Effect**: Warning; extra values may be dropped or placed in an unnamed column.

2. `read_csv("a,b,c\n1,2\n1,2,3,4")`
- **Issue**: Inconsistent row lengths (missing and extra values).
- **Effect**: Warning; missing values filled with NA, extra values might create an extra column.

3. `read_csv("a,b\n\"1")`
- **Issue**: Unterminated quoted string.
- **Effect**: Parsing error.

4. `read_csv("a,b\n1,2\na,b")`
- **Issue**: Data row (a,b) repeats column names, causing type mismatch.
- **Effect**: Warning; numeric columns converted to character.

5. `read_csv("a;b\n1;3")`
- **Issue**: Wrong delimiter (; instead of ,).
- **Effect**: Read as a single column; **Fix**: Use read_delim(..., delim = ";").

6. Practice referring to non-syntactic names in the following data frame by:

1. Extracting the variable called 1.
- Annoying[["1"]] or annoying$`1`

2. Plotting a scatterplot of 1 vs. 2.
- plot(annoying$`1`, annoying$`2`, main = "Scatterplot of 1 vs 2", xlab = "1", ylab = "2")

3. Creating a new column called 3, which is 2 divided by 1.

- annoying$`3` <- annoying$`2` / annoying$`1`

4. Renaming the columns to one, two, and three.

- colnames(annoying) <- c("one", "two", "three")

Executables

```
#Chapter 7  Data import .........................

#7.1.1 Prerequisites

library(tidyverse)

#7.2 Reading data from a file

students <- read_csv("data/students.csv")

students <- read_csv("https://pos.it/r4ds-students-csv")

#7.2.1 Practical advice .......................

students

students <- read_csv("/Users/anoushkagurung.csv", na = c("N/A", ""))

students

#Student ID contains a space and requires backticks use

students |>
  rename(
    student_id = `Student ID`,
    full_name = `Full Name`
  )

students |> janitor::clean_names()
students |>
  janitor::clean_names() |>
  mutate(meal_plan = factor(meal_plan))

students <- students |>
  janitor::clean_names() |>
  mutate(
    meal_plan = factor(meal_plan),
    age = parse_number(if_else(age == "five", "5", age))
  )

students

#7.2.2 Other arguments ................................

#read_csv() can read text strings that you’ve created and formatted like a CSV file:

read_csv(
  "a,b,c
  1,2,3
  4,5,6"
)

read_csv(
  "The first line of metadata
  The second line of metadata
  x,y,z
  1,2,3",
  skip = 2
)

read_csv(
  "# A comment I want to skip
  x,y,z
  1,2,3",
  comment = "#"
)

read_csv(
  "1,2,3
  4,5,6",
  col_names = FALSE
)

read_csv(
  "1,2,3
  4,5,6",
  col_names = c("x", "y", "z")
)

#7.2.3 Other file types ..............................

# read_csv2() reads semicolon-separated files. These use ; instead of , to separate fields and are
common in countries that use , as the decimal marker.
#
# read_tsv() reads tab-delimited files.
#
# read_delim() reads in files with any delimiter, attempting to automatically guess the delimiter if
you don't specify it.
#
# read_fwf() reads fixed-width files. You can specify fields by their widths with fwf_widths() or
by their positions with fwf_positions().
#
# read_table() reads a common variation of fixed-width files where columns are separated by
white space.
#
# read_log() reads Apache-style log files.

#7.3.1 Guessing types ............

read_csv("
  logical,numeric,date,string
  TRUE,1,2021-01-15,abc
  false,4.5,2021-02-15,def
  T,Inf,2021-02-16,ghi
")

#7.3.2 Missing values, column types, and problems ................
```

```
simple_csv <- "
  x
  10
  .
  20
  30"

read_csv(simple_csv)

df <- read_csv(
  simple_csv,
  col_types = list(x = col_double())
)

problems(df)

read_csv(simple_csv, na = ".")  #if '.' found, change to N/A

#7.3.3 Column types ................................

another_csv <- "
x,y,z
1,2,3"

read_csv(
  another_csv,
  col_types = cols(.default = col_character())
)

read_csv(
  another_csv,
  col_types = cols_only(x = col_character())
)

#7.4 Reading data from multiple files ...................................

sales_files <- c("data/01-sales.csv", "data/02-sales.csv", "data/03-sales.csv")
read_csv(sales_files, id = "file")

sales_files <- c(
  "https://pos.it/r4ds-01-sales",
  "https://pos.it/r4ds-02-sales",
  "https://pos.it/r4ds-03-sales"
)
read_csv(sales_files, id = "file")

sales_files <- list.files("data", pattern = "sales\\.csv$", full.names = TRUE)
sales_files
#> [1] "data/01-sales.csv" "data/02-sales.csv" "data/03-sales.csv"

#7.5 Writing to a file ............................

write_csv(students, "students.csv")

students

write_csv(students, "students-2.csv")
read_csv("students-2.csv")

write_rds(students, "students.rds")
read_rds("students.rds")

library(arrow)
write_parquet(students, "students.parquet")
read_parquet("students.parquet")

#7.6 Data entry

tibble(
  x = c(1, 2, 5),
  y = c("h", "m", "g"),
  z = c(0.08, 0.83, 0.60)
)

tribble(
  ~x, ~y, ~z,
  1, "h", 0.08,
  2, "m", 0.83,
  5, "g", 0.60
)
```

#Chapter 8 Workflow: getting help

#8.2 Making a reprex