Notes on Ch 7: Data Import

N. Lim

2025-07-01

Prerequisites

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

Reading data from file

- using read_csv:

```
students <- read_csv("https://pos.it/r4ds-students-csv", na = c("N/A", ""))
```

```
## Rows: 6 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): Full Name, favourite.food, mealPlan, AGE
## dbl (1): Student ID
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
students
```

```
## # A tibble: 6 x 5
##   `Student ID` `Full Name`      favourite.food     mealPlan            AGE
##          <dbl> <chr>            <chr>              <chr>               <chr>
## 1            1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
## 2            2 Barclay Lynn     French fries       Lunch only          5
## 3            3 Jayendra Lyne    <NA>               Breakfast and lunch 7
## 4            4 Leon Rossini     Anchovies          Lunch only          <NA>
## 5            5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
## 6            6 Güvenç Attila    Ice cream          Lunch only          6
```

- Getting rid of space/s in the column names:

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

```
## # A tibble: 6 x 5
##   student_id full_name        favourite.food     mealPlan            AGE
##        <dbl> <chr>            <chr>              <chr>               <chr>
## 1          1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
## 2          2 Barclay Lynn     French fries       Lunch only          5
## 3          3 Jayendra Lyne    <NA>               Breakfast and lunch 7
## 4          4 Leon Rossini     Anchovies          Lunch only          <NA>
## 5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
## 6          6 Güvenç Attila    Ice cream          Lunch only          6
```

Using `janitor::clean_names()` to convert column names to snake case:

```
students |>
  janitor::clean_names()
```

```
## # A tibble: 6 x 5
##   student_id full_name        favourite_food     meal_plan           age
##        <dbl> <chr>            <chr>              <chr>               <chr>
## 1          1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
## 2          2 Barclay Lynn     French fries       Lunch only          5
## 3          3 Jayendra Lyne    <NA>               Breakfast and lunch 7
## 4          4 Leon Rossini     Anchovies          Lunch only          <NA>
## 5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
## 6          6 Güvenç Attila    Ice cream          Lunch only          6
```

Changing column type to factor:

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

```
## # A tibble: 6 x 5
##   student_id full_name        favourite_food     meal_plan           age
##        <dbl> <chr>            <chr>              <fct>               <chr>
## 1          1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
## 2          2 Barclay Lynn     French fries       Lunch only          5
## 3          3 Jayendra Lyne    <NA>               Breakfast and lunch 7
## 4          4 Leon Rossini     Anchovies          Lunch only          <NA>
## 5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
## 6          6 Güvenç Attila    Ice cream          Lunch only          6
```

Fixing the `age` column: first convert "five" to "5" then parse them into numbers using `parse_number()`

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

students
```

```
## # A tibble: 6 x 5
##   student_id full_name        favourite_food     meal_plan             age
##        <dbl> <chr>            <chr>              <fct>               <dbl>
## 1          1 Sunil Huffmann   Strawberry yoghurt Lunch only              4
## 2          2 Barclay Lynn     French fries       Lunch only              5
## 3          3 Jayendra Lyne    <NA>               Breakfast and lunch     7
## 4          4 Leon Rossini     Anchovies          Lunch only             NA
## 5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch     5
## 6          6 Güvenç Attila    Ice cream          Lunch only              6
```

Other arguments

- `read_csv` can red text stings formatted like a CSV file:

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

```
## Rows: 2 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): a, b, c
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 2 x 3
##       a     b     c
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     4     5     6
```

- by default, the first line is used as column header. In case the first line/s in the CSV is/are not really the header, we can skip it/them using `skip = n`:

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

```
## Rows: 1 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): x, y, z
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 1 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1     1     2     3
```

Skipping comments using `comment =`:

```r
read_csv(
  "# A comment that needs to be skipped
  x, y, z
  1, 2, 3",
  comment = "#"
)
```

```
## Rows: 1 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): x, y, z
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 1 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1     1     2     3
```

If there are no column names:

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

```
## Rows: 2 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): X1, X2, X3
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 2 x 3
##      X1    X2    X3
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     4     5     6
```

or we can pass our own column names using `col_names =`:

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

```
## Rows: 2 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): x, y, z
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## # A tibble: 2 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     4     5     6
```

Reading other file types

- `read_csv2()`: for semicolon-separated files. Common in countries that use `,` as a decimal marker.

- `read_tsv()`: reads tab-delimited data.

- `read_delim()`: reads files with any delimeter. It attempts to automatically guess the delimeter if you don't specify it.

- `read_fwf()`: reads fixed-width files.

- `read_table()`: reads a common varialtion of fixed-width files where columns are separated by white space.

- `read_log()`: reads Apache-style log files.

Guessing column types

- **readr** uses a heuristic to figure out the column types.
- in real-world data, even this clever heuristic will fail to correctly guess the correct data type, especially if the data is "dirty"

Missing values, column types, problems

- R will "coerce" the column type to what it deems "best" when data is incomplete (or if it has missing values) – my words, not the book's authors.

Example:

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

read_csv(simple_csv)
```

```
## Rows: 4 Columns: 1
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): x
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 4 x 1
##   x
##   <chr>
## 1 10
## 2 .
## 3 20
## 4 30
```

Since the column data contains a dot, R coerced the column type to be 'character' even though most of the content is numerical. What if specify the data type?

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

```
## Warning: One or more parsing issues, call `problems()` on your data frame for details,
## e.g.:
##   dat <- vroom(...)
##   problems(dat)
```

We can see that R "complains"... We can see the problem using the `problems()` function:

```
problems(df)
```

```
## # A tibble: 1 x 5
##     row   col expected actual file
##   <int> <int> <chr>    <chr>  <chr>
## 1     3     1 a double .      /tmp/RtmprIYoNi/file8eda4755b2d36
```

Correcting the problem:

```
read_csv(
  simple_csv,
  na = "."
)
```

```
## Rows: 4 Columns: 1
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (1): x
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 4 x 1
##       x
##   <dbl>
## 1    10
## 2    NA
## 3    20
## 4    30
```

Column types

Column types provided by readr: - `col_logical()` and `col_double()`: used to read logicals and real numbers. Rarely needed since readr will usually get them correctly.
- `col_integer()`: reads integers. Somethimes useful because integers occupy less memory than doubles.
- `col_character()`: reads strings.
- `col_factor()` or `col_date()`, and `col_datetime()`: used to create factors, dates, and date-times. - `col_number()`: ignores non-numeric components; useful when dealing with currencies. - `col_skip()`: skips a column so it's not included in the result. Used to save time when you have a large CSV file and am interested only in some of the columns.

Whil R will usually correctly guess the column type, it is possible to override this by using `cols(.default = <column type()>)`

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

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

```
## # A tibble: 1 x 3
##   x     y     z
##   <chr> <chr> <chr>
## 1 1     2     3
```

Using `cols_only()` to specify which columns to read:

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

```
## # A tibble: 1 x 1
##   x
##   <chr>
## 1 1
```

Reading data from multiple files

- Sometimes, data is split across multiple files instead of being contained in a single file. We can read multiple files by using the concatenate function `c()`. This will read all the files and stack them on top of each other:

```
sales_files <- c("file1.csv", "file2.csv", "file3.csv)

read_csv(sales_files, id = "file")
```

- Reading multiple urls:

```
sales_files <- c(
  "<URL 1>",
  "<URL 2>",
  "<URL 3>"
)

read_csv(sales_files, id = "file")
```

- the argument `id = "file"` will add a new column called `file` to the resulting data frame. This will contain filename where the data came from.
- Reading multiple files when there are may of them:

```
sales_files <- list.files("data", pattern = "sales\\.csv$", full.names = TRUE)

sales_files
```

Writing to file

- Writing to csv:

```r
write_csv(students, "students.csv")
```

- For writing to tsv, use `write_tsv()`.
- When writing files to tsv and csv, the variable type will be lost.
- When you read in the CSV or TSV files that you saved, you will start over from a plain text file.
- To preserve the information from parsing, we can save the data using `write_rds()`. This will store data in a custom binary format in R.
- Likewise, we can use `read_rds()` to read in the RDS file that we have created.
- We can also save the files to parquet format using the **arrow** library which contains the functions `write_parquet()` and `read_parquet()`.

Entering data manually

- Using the `tibble()` function:

```r
tibble(
  x = c(1, 2, 5),
  y = c("h", "m", "g"),
  z = c(0.08, 0.83, 0.69)
)
```

```
## # A tibble: 3 x 3
##       x y         z
##   <dbl> <chr> <dbl>
## 1     1 h      0.08
## 2     2 m      0.83
## 3     5 g      0.69
```

- Using the `tribble()` function:

```r
tribble(
  ~x, ~y, ~z,
  1, "h", 0.08,
  2, "m", 0.83,
  5, "g", 0.69
)
```

```
## # A tibble: 3 x 3
##       x y         z
##   <dbl> <chr> <dbl>
## 1     1 h      0.08
## 2     2 m      0.83
## 3     5 g      0.69
```