Data frame accessing and piping on R

Outline

- Data frame accessing.
- Pipe operator %>%.

Learning Objectives

1. Learn how to access columns of a data frame and tibble.
2. Learn how to use the pipe operator %>%.

Data frame accessing

Data frame accessing through `mtcars` example

- Built-in dataset:

```
mtcars
```

- Extracted from the 1974 Motor Trend US magazine.
- Comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).
- Dimension 32 by 11.

Data frame accessing through `mtcars` example

cont'd

- When you type `mtcars` in your console, it tries to display the whole dataset.
- Depending on the resolution of your screen, the output format may vary, and it is sometimes hard to make good sense of the data by looking at the output on your screen.
- Instead, try typing the following:

```
library(tibble)
as_tibble(mtcars)
```

Data frame accessing through `mtcars` example

cont'd

```
# A tibble: 32 x 11
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1  21       6  160    110  3.9   2.62  16.5     0     1     4     4
 2  21       6  160    110  3.9   2.88  17.0     0     1     4     4
 3  22.8     4  108     93  3.85  2.32  18.6     1     1     4     1
 4  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
 5  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2
 6  18.1     6  225    105  2.76  3.46  20.2     1     0     3     1
 7  14.3     8  360    245  3.21  3.57  15.8     0     0     3     4
 8  24.4     4  147.    62  3.69  3.19  20       1     0     4     2
 9  22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2
10  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
# ... with 22 more rows
```

Data frame accessing through `mtcars` example

cont'd

- The function `as_tibble` converts the data to a *tibble*.
- It is different from data frames in a few ways:
  1. When printing a data frame, it does not print all the rows and all the columns. This makes it better for inspecting a data frame.
  2. The output is colour coded, allowing us to distinguish positive and negative values quickly.
  3. Extraneous decimal places are suppressed, allowing for quicker scanning of the data frame.

```
tibble(-2909,1507,0.610,0.2216)
```

```
# A tibble: 1 x 4
  `-2909` `1507` `0.61` `0.2216`
    <dbl>  <dbl>  <dbl>    <dbl>
1   -2909   1507   0.61    0.222
```

Data frame accessing through `mtcars` example

cont'd

- Earlier, we saw that `mtcars` contain columns such as `mpg`, `cyl`, `disp`, etc..
- How do we access the column we need?
- Recall use the $ symbol.
- The syntax is <dataframe_name>$<column_name>.

```
mtcars$mpg
```

```
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
[16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
[31] 15.0 21.4
```

Data frame accessing through `mtcars` example

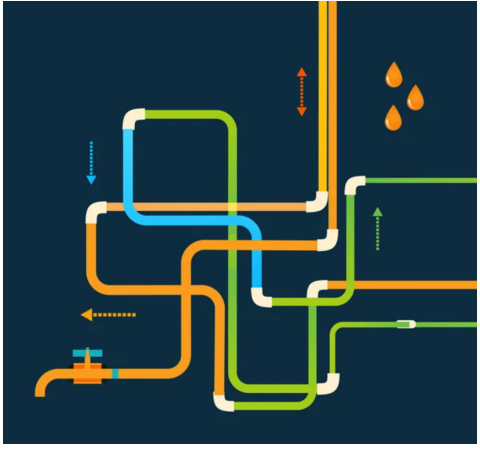
cont'd

- How do we divide values of two columns, say `hp` over `wt`?
- Just use the / sign appropriately.

```
mtcars$hp / mtcars$wt
```

```
 [1] 41.98473 38.26087 40.08621 34.21462 50.87209 30.34682 68.62745 19.43574
 [9] 30.15873 35.75581 35.75581 44.22604 48.25737 47.61905 39.04762 39.63864
[17] 43.03087 30.00000 32.19814 35.42234 39.35091 42.61364 43.66812 63.80208
[25] 45.51365 34.10853 42.52336 74.68605 83.28076 63.17690 93.83754 39.20863
```

Pipe operator %>%



Pipe operator %>%

- The pipe operator %>% is an operator defined by the `magrittr` package. It is loaded automatically by the `simmer` package.
- Behind the scenes, it converts
  - `x %>% f(y)` into `f(x, y)`;
  - `x %>% f(y) %>% g(z)` into `f(x,y) %>% g(z)`, which is just `g(f(x,y), z)`.
- It is neater to *pipe* the output this way.
  - Not required to name and store any intermediate dataset created (for example `f(x,y)` in `g(f(x,y), z)` for `x %>% f(y) %>% g(z)`).

Pipe Operator %>%

cont'd

- In an upcoming video, you will be exposed to a *bank* simulation and our goal is to *run* the simulation until 1000 units of time.
- The code fragment looks like:

```
bank %>% run(until = 1000)
```

- This is equivalent to:

```
run(bank, until = 1000)
```

Pipes and Readable Code

- The goal of the pipe syntax is to make code more readable, not to make code shorter.
- Refrain from code such as this:

```
trajectory %>% seize( ...  ) %>% timeout( ... ) %>% release( ... )
%>% log_( ... )
```

- Instead, put at most one pipe operator per line:

```
trajectory %>%
  seize( ...  ) %>%
  timeout( ... ) %>%
  release( ... ) %>%
  log_( ... )
```

- That makes your code much more team-friendly.

Pipes and Readable Code

cont'd

- The pipe syntax saves you the trouble of storing intermediate variables too.
- You are not required to do the following:

```
temp1 <- seize(trajectory, ... )
temp2 <- timeout(temp1, ... )
temp3 <- release(temp2, ... )
temp4 <- log_(temp3, ... )
```

- Instead, we pipe sequentially.

```
trajectory %>%
  seize( ...  ) %>%
  timeout( ... ) %>%
  release( ... ) %>%
  log_( ... )
```

Summary

In this video, we have:

1. Learned how to access columns of a data frame and tibble.
2. Learned how to use the pipe operator %>%.