SOC722 Exercise Week 1

Zhe Chen

2026-01-18

Exercise 1.1

Try typing `typeof(mtcars)` and `class(mtcars)` in the console to see what happens.

Now type the following chunks of code into your console and understand what they do:

- `nrow(mtcars)`
- `ncol(mtcars)`
- `length(mtcars)`
- `dim(mtcars)`
- `rownames(mtcars)`
- `colnames(mtcars)`

Briefly describe what each of these do.

```
typeof(mtcars)
```

```
[1] "list"
```

- The type of `mtcars` is `list`.

```
class(mtcars)
```

```
[1] "data.frame"
```

- The class of `mtcars` is `data.frame`.

```r
nrow(mtcars)
```

```
[1] 32
```

- The number of rows in `mtcars` is 32.

```r
ncol(mtcars)
```

```
[1] 11
```

- The number of columns in `mtcars` is 11.

```r
length(mtcars)
```

```
[1] 11
```

- The length of `mtcars` is 11.

```r
dim(mtcars)
```

```
[1] 32 11
```

- The dimension of `mtcars` is (32,11), i.e., it has 32 rows and 11 columns.

```r
rownames(mtcars)
```

```
 [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
 [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
 [7] "Duster 360"          "Merc 240D"           "Merc 230"
[10] "Merc 280"            "Merc 280C"           "Merc 450SE"
[13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwood"
[16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
[19] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"
[22] "Dodge Challenger"    "AMC Javelin"         "Camaro Z28"
[25] "Pontiac Firebird"    "Fiat X1-9"           "Porsche 914-2"
[28] "Lotus Europa"        "Ford Pantera L"      "Ferrari Dino"
[31] "Maserati Bora"       "Volvo 142E"
```

- The names of rows (observations) in `mtcars` are “Mazda Rx4”, “Mazda Rx4 Wag”, “Datsun 710”...

```
colnames(mtcars)
```

```
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
[11] "carb"
```

- The names of columns (variables) in `mtcars` are “mpg”, “cyl”, “disp”…

Exercise 1.2

I suggest you always use long-form when creating logical vectors. Try assigning a different value to `TRUE` and to `T`.

Code

```
T <- 123
TRUE <- 123
```

What just happened?

```
T <- 123
T
```

```
[1] 123
```

```
TRUE <- 123
```

```
Error in TRUE <- 123: invalid (do_set) left-hand side to assignment
```

```
TRUE
```

```
[1] TRUE
```

- Assign value 123 to `T`. It succeeds.
- Assign value 123 to `TRUE`. It fails, and R returns `Error in TRUE <- 123 : invalid (do_set) left-hand side to assignment`.

Exercise 1.3

Implicit coercion

You can create atomic vectors of any length with `c()` for "concatenate".

For example:

```
lgl <- c(TRUE, FALSE, NA)
int <- c(1L, 6L, NA, 10L)
dbl <- c(1, NA, 2.5, 4.5)
chr <- c(NA, "these are", "some strings")
```

Recall that atomic vectors are homogeneous. If you try to concatenate vectors of different types you will end up discovering **implicit coercion**. Basically, different types will be coerced in the following order: logical → integer → double → character. For example, a logical and a character combine into a character:

```
str(c(TRUE, "chr")) ## str() is (almost) identical to dplyr::glimpse()
```

```
 chr [1:2] "TRUE" "chr"
```

Test your knowledge of the vector coercion rules by predicting the output of the following uses of `c()`:

```
c(1, FALSE)
c("a", 1)
c(TRUE, 1L)
```

- `c(1, FALSE)` is a *double*.
- `c("a", 1)` is a *character*.
- `c(TRUE, 1L)` is an *integer*.

Exercise 1.4

Explicit coercion

Explicit coercion happens when you call a function like `as.logical()`, `as.integer()`, `as.double()`, or `as.character()`. Use `as.integer()` on FALSE and TRUE, what values do they get coerced to?

```
as.integer(FALSE)
```

```
[1] 0
```

- The value of `as.integer(FALSE)` is 0.

```
as.integer(TRUE)
```

```
[1] 1
```

- The value of `as.integer(TRUE)` is 1.

Exercise 1.5

The most common form of implicit coercion

The following chunk of code creates a logical vector of size 75.

```
x <- sample(c(TRUE, FALSE), size = 75, replace = TRUE)
str(x)
```

```
 logi [1:75] FALSE FALSE TRUE TRUE TRUE TRUE ...
```

```
x <- sample(c(TRUE, FALSE), size = 75, replace = TRUE)
```

- The number of `TRUE` in `x` is:

```
sum(x)
```

```
[1] 39
```

- The proportion of `TRUE` in `x` is:

```
mean(x)
```

```
[1] 0.52
```

```
mean(x) == sum(x) / length(x)
```

```
[1] TRUE
```

- Yes, `mean(x)` and `sum(x) / length(x)` are equal.

Exercise 1.6

What is the difference between mtcars[“mpg”] and mtcars[[“mpg”]]? More generally, what is the difference between the [and [[operators?

Which of the following two is `TRUE`? “‘ identical(mtcars[“mpg”], mtcars$mpg) $identical(mtcars[["mpg"]], mtc$

```
mtcars["mpg"]
```

```
                   mpg
Mazda RX4         21.0
Mazda RX4 Wag     21.0
Datsun 710        22.8
Hornet 4 Drive    21.4
Hornet Sportabout 18.7
Valiant           18.1
Duster 360        14.3
Merc 240D         24.4
Merc 230          22.8
Merc 280          19.2
Merc 280C         17.8
Merc 450SE        16.4
Merc 450SL        17.3
Merc 450SLC       15.2
Cadillac Fleetwood  10.4
Lincoln Continental 10.4
Chrysler Imperial   14.7
Fiat 128          32.4
Honda Civic       30.4
Toyota Corolla    33.9
Toyota Corona     21.5
Dodge Challenger  15.5
AMC Javelin       15.2
Camaro Z28        13.3
Pontiac Firebird  19.2
Fiat X1-9         27.3
Porsche 914-2     26.0
Lotus Europa      30.4
Ford Pantera L    15.8
Ferrari Dino      19.7
Maserati Bora     15.0
Volvo 142E        21.4
```

```
str(mtcars["mpg"])
```

```
'data.frame':   32 obs. of  1 variable:
 $ mpg: num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
```

```
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
```

```
str(mtcars[["mpg"]])
```

```
 num [1:32] 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
```

- `[]` returns a data frame, while `[[]]` returns a vector (i.e., the content of `mpg` column in that data frame)

```
identical(mtcars["mpg"], mtcars$mpg)
```

```
[1] FALSE
```

```
identical(mtcars[["mpg"]], mtcars$mpg)
```

```
[1] TRUE
```

- `identical(mtcars[["mpg"]], mtcars$mpg)` is `TURE`.

Exercise 1.7

> `letters` is a built-in object in R that contains the 26 letters of English alphabet.
>
> Using the [operator, do the following: - Extract the 17th value of `letters` - Create a sequence of even numbers from 2 to 26 and use that to subset `letters` - Use 8:12 to subset `letters`.
>
> This is known as integer subsetting.
>
> What happens if instead of [you use [[?

```
letters
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

```
letters[17]
```

```
[1] "q"
```

```
even_numbers <- seq(2, 26, 2)
letters[even_numbers]
```

```
 [1] "b" "d" "f" "h" "j" "l" "n" "p" "r" "t" "v" "x" "z"
```

```
letters[8:12]
```

```
[1] "h" "i" "j" "k" "l"
```

```
letters[[8:12]]
```

```
Error in letters[[8:12]]: attempt to select more than one element in vectorIndex
```

- If using `[[8:12]]`, R returns `Error in letters[[8:12]] : attempt to select more than one element in vectorIndex`. Because `[[]]` only allows for extracting one element.

Exercise 1.8

Now that you know all this

Replace the 18th value of `letters` with a missing value (`NA`).

```
letters[18] <- NA
letters
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" NA  "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

Exercise 1.9

Subset `mtcars` so that we only see the observations for which `cyl == 4`.

Subset `mtcars` so that we only see the observations for which `mpg` is greater than 23.

```
mtcars[mtcars$cyl == 4, ]
```

```
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
mtcars[mtcars$mpg > 23, ]
```

```
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
```

Exercise 1.10

Using what I told you earlier about the `search()` function, explain why you get two different errors. What is going on? What is R doing when you type `table(year)`? (You might want to type `search()` into the console again). In what package does R find the year object?

```r
table(year)
```

```
Error: object 'year' not found
```

```r
search()
```

```
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

- R returns `Error: object 'year' not found`. It is because neither `palmerpenguins` or `tidyverse` is loaded, and R cannot find the object `year` in the global environment.

```r
library(tidyverse)
```

```
Warning: package 'purrr' was built under R version 4.5.2
```

```
-- 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   3.5.2     v tibble    3.3.0
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.2.0
-- 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 beco
```

```r
library(palmerpenguins)
```

```
Attaching package: 'palmerpenguins'

The following objects are masked from 'package:datasets':

    penguins, penguins_raw
```

```
table(year)
```

```
Error in unique.default(x, nmax = nmax): unique() applies only to vectors
```

```
search()
```

```
 [1] ".GlobalEnv"             "package:palmerpenguins" "package:lubridate"
 [4] "package:forcats"        "package:stringr"        "package:dplyr"
 [7] "package:purrr"          "package:readr"          "package:tidyr"
[10] "package:tibble"         "package:ggplot2"        "package:tidyverse"
[13] "package:stats"          "package:graphics"       "package:grDevices"
[16] "package:utils"          "package:datasets"       "package:methods"
[19] "Autoloads"              "package:base"
```

```
find("year")
```

```
[1] "package:lubridate"
```

```
year
```

```
function (x)
{
    UseMethod("year")
}
<bytecode: 0x0000028848ebe120>
<environment: namespace:lubridate>
```

- R returns `Error in unique.default(x, nmax = nmax): unique() applies only to vectors`. It is because R finds `year` in `tidyverse` which is a `function`, and R cannot apply `unique()` to a `function`.

Exercise 1.11

Use `slice()` to extract the even-numbered rows in the penguins dataset.

It will look something like this:

```
penguins |>
slice("SOME NUMERIC VECTOR GOES HERE")
```

```
nrow(penguins)
```

```
[1] 344
```

```
penguins |>
  slice(seq(2, 344, 2))
```

```
# A tibble: 172 x 8
   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
   <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
 1 Adelie  Torgersen           39.5          17.4               186        3800
 2 Adelie  Torgersen           NA            NA                  NA          NA
 3 Adelie  Torgersen           39.3          20.6               190        3650
 4 Adelie  Torgersen           39.2          19.6               195        4675
 5 Adelie  Torgersen           42            20.2               190        4250
 6 Adelie  Torgersen           37.8          17.3               180        3700
 7 Adelie  Torgersen           38.6          21.2               191        3800
 8 Adelie  Torgersen           36.6          17.8               185        3700
 9 Adelie  Torgersen           42.5          20.7               197        4500
10 Adelie  Torgersen           46            21.5               194        4200
# i 162 more rows
# i 2 more variables: sex <fct>, year <int>
```

Now use `slice()` to extract every third row—i.e., row 3, 6, 9, and so on.

```
penguins |>
  slice(seq(3, 344, 3))
```

```
# A tibble: 114 x 8
   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
   <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
 1 Adelie  Torgersen           40.3          18                 195        3250
 2 Adelie  Torgersen           39.3          20.6               190        3650
 3 Adelie  Torgersen           34.1          18.1               193        3475
 4 Adelie  Torgersen           37.8          17.3               180        3700
 5 Adelie  Torgersen           34.6          21.1               198        4400
 6 Adelie  Torgersen           42.5          20.7               197        4500
 7 Adelie  Biscoe              37.8          18.3               174        3400
 8 Adelie  Biscoe              38.2          18.1               185        3950
 9 Adelie  Biscoe              40.6          18.6               183        3550
```

```
10 Adelie  Biscoe              40.5          18.9                180        3950
# i 104 more rows
# i 2 more variables: sex <fct>, year <int>
```

Exercise 1.12

Use `filter()` to extract the observations in the penguins dataset for which `species == "Gentoo"`, `island == "Biscoe"`, and `body_mass_g` is between 5,000 and 5,500.

```
penguins |>
  filter(species == "Gentoo", island == "Biscoe") |>
  filter(body_mass_g >= 5000, body_mass_g <= 5500)
```

```
# A tibble: 39 x 8
   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
   <fct>   <fct>           <dbl>         <dbl>             <int>       <int>
 1 Gentoo  Biscoe           47.6          14.5               215        5400
 2 Gentoo  Biscoe           46.7          15.3               219        5200
 3 Gentoo  Biscoe           46.8          15.4               215        5150
 4 Gentoo  Biscoe           48.7          15.1               222        5350
 5 Gentoo  Biscoe           45.1          14.5               215        5000
 6 Gentoo  Biscoe           46.3          15.8               215        5050
 7 Gentoo  Biscoe           42.9          13.1               215        5000
 8 Gentoo  Biscoe           46.1          15.1               215        5100
 9 Gentoo  Biscoe           47.3          15.3               222        5250
10 Gentoo  Biscoe           45.1          14.5               207        5050
# i 29 more rows
# i 2 more variables: sex <fct>, year <int>
```