**תרגיל 1.1 ופתרונותיו:**

1. חשב את הממוצע של כל עמודה ב- `mtcars.

output <- vector("double", ncol(mtcars))

names(output) <- names(mtcars)

*# loop over the names of mtcars*

**for** (i **in** names(mtcars)) {

output[i] <- mean(mtcars[[i]])

}

output

***## mpg cyl disp hp drat wt qsec***

***## 20.090625 6.187500 230.721875 146.687500 3.596563 3.217250 17.848750***

***## vs am gear carb***

***## 0.437500 0.406250 3.687500 2.812500***

*# without names*

**for**(i **in** seq\_along(mtcars)){

output[i]<- mean(mtcars[[i]])

}

names(output) <- names(mtcars)

output

***## mpg cyl disp hp drat wt qsec***

***## 20.090625 6.187500 230.721875 146.687500 3.596563 3.217250 17.848750***

***## vs am gear carb***

***## 0.437500 0.406250 3.687500 2.812500***

2. ברר מהו סוג העמודה עבור כל עמודה בטבלה `nycflights13::flights`

output <- vector("list", ncol(nycflights13::flights))

names(output) <- names(nycflights13::flights)

**for** (i **in** names(nycflights13::flights)){

output[[i]] <- class(nycflights13::flights[[i]])

}

output

***## $year***

***## [1] "integer"***

***##***

***## $month***

***## [1] "integer"***

***##***

***## $day***

***## [1] "integer"***

***##***

***## $dep\_time***

***## [1] "integer"***

***##***

***## $sched\_dep\_time***

***## [1] "integer"***

***##***

***## $dep\_delay***

***## [1] "numeric"***

***##***

***## $arr\_time***

***## [1] "integer"***

***##***

***## $sched\_arr\_time***

***## [1] "integer"***

***##***

***## $arr\_delay***

***## [1] "numeric"***

***##***

***## $carrier***

***## [1] "character"***

***##***

***## $flight***

***## [1] "integer"***

***##***

***## $tailnum***

***## [1] "character"***

***##***

***## $origin***

***## [1] "character"***

***##***

***## $dest***

***## [1] "character"***

***##***

***## $air\_time***

***## [1] "numeric"***

***##***

***## $distance***

***## [1] "numeric"***

***##***

***## $hour***

***## [1] "numeric"***

***##***

***## $minute***

***## [1] "numeric"***

***##***

***## $time\_hour***

***## [1] "POSIXct" "POSIXt"***

*# The class of an object can have multiple values. For example, the class of the time\_hour column is POSIXct, POSIXt. therefore we used list an not a vector.*

3. חשב את מספר הערכים הייחודיים בכל עמודה בטבלה `iris`.

data("iris")

iris\_uniq <- vector("double", ncol(iris))

names(iris\_uniq) <- names(iris) *# we'll talk aboput it soon, I'll loop over names of the df.*

**for** (i **in** names(iris)) {

iris\_uniq[i] <- n\_distinct(iris[[i]])

}

iris\_uniq

***## Sepal.Length Sepal.Width Petal.Length Petal.Width Species***

***## 35 23 43 22 3***

4. זה נפוץ לראות לופ שלא מקצה מקום לתוצאות מלכתחילה, אלא מגדיל את הווקטור בכל איטראציה:

output <- vector("integer", 0)

**for** (i **in** seq\_along(x)) {

output <- c(output, lengths(x[[i]]))

}

output

כיצד זה משפיע על המהירות? תכנון ובצע ניסוי שבודק זאת.

*# define two functions:*

*# 1. add\_to\_vector will append to a vector.*

*# 2.add\_to\_vector\_2 pre-allocates a vector.*

add\_to\_vector <- **function**(n) {

output <- vector("integer", 0)

**for** (i **in** seq\_len(n)) {

output <- c(output, i)

}

output

}

add\_to\_vector\_2 <- **function**(n) {

output <- vector("integer", n)

**for** (i **in** seq\_len(n)) {

output[[i]] <- i

}

output

}

add\_to\_vector\_2(12)

***## [1] 1 2 3 4 5 6 7 8 9 10 11 12***

*# The package microbenchmark run expressions several times to compare the time it takes. The microbenchmark() function will run an R expression a number of times and time it.*

library("microbenchmark")

timings <- microbenchmark(add\_to\_vector(10000), add\_to\_vector\_2(10000), times = 10)

timings

***## Unit: microseconds***

***## expr min lq mean median uq max***

***## add\_to\_vector(10000) 108550.2 115178.5 131815.24 126324.05 136059.1 207251.2***

***## add\_to\_vector\_2(10000) 412.9 477.9 891.56 640.45 1433.3 1521.5***

***## neval***

***## 10***

***## 10***

**תרגיל 2.7 ופתרונותיו**

1. יש לכם תיקייה מלאה בקבצי CSV שאתם רוצים לעלות ל-R. ה-`path` לכל אחד מהקבצים נמצא בווקטור:

dir("8\_data", pattern = "\\.csv$", full.names = TRUE)

כתבו `for loop` שייבא את הקבצים ויהפוך אותם ל-`df` יחיד.

files <- dir("8\_data/", pattern = "\\.csv", full.names = TRUE)

files

***## [1] "8\_data/file1.csv" "8\_data/file2.csv" "8\_data/file3.csv"***

df\_list <- vector("list", length(files))

**for** (i **in** seq\_along(files)) {

df\_list[[i]] <- read\_csv(files[[i]], locale = locale(encoding = "windows-1255"))

}

***## Parsed with column specification:***

***## cols(***

***## id = col\_double(),***

***## var1 = col\_double(),***

***## var2 = col\_character(),***

***## var3 = col\_character()***

***## )***

***## Parsed with column specification:***

***## cols(***

***## id = col\_double(),***

***## var1 = col\_double(),***

***## var2 = col\_character(),***

***## var3 = col\_character()***

***## )***

***## Parsed with column specification:***

***## cols(***

***## id = col\_double(),***

***## var1 = col\_double(),***

***## var2 = col\_character(),***

***## var3 = col\_character()***

***## )***

df\_list

***## [[1]]***

***## # A tibble: 5 x 4***

***## id var1 var2 var3***

***## <dbl> <dbl> <chr> <chr>***

***## 1 20 0 a המבורגר***

***## 2 21 1 b בננה***

***## 3 22 2 c גזר***

***## 4 23 3 d דבש***

***## 5 24 4 e אגס***

***##***

***## [[2]]***

***## # A tibble: 3 x 4***

***## id var1 var2 var3***

***## <dbl> <dbl> <chr> <chr>***

***## 1 25 5 a המבורגר***

***## 2 26 6 b בננה***

***## 3 27 7 c גזר***

***##***

***## [[3]]***

***## # A tibble: 3 x 4***

***## id var1 var2 var3***

***## <dbl> <dbl> <chr> <chr>***

***## 1 28 8 a המבורגר***

***## 2 29 9 b בננה***

***## 3 30 10 c גזר***

bind\_rows(df\_list)

***## # A tibble: 11 x 4***

***## id var1 var2 var3***

***## <dbl> <dbl> <chr> <chr>***

***## 1 20 0 a המבורגר***

***## 2 21 1 b בננה***

***## 3 22 2 c גזר***

***## 4 23 3 d דבש***

***## 5 24 4 e אגס***

***## 6 25 5 a המבורגר***

***## 7 26 6 b בננה***

***## 8 27 7 c גזר***

***## 9 28 8 a המבורגר***

***## 10 29 9 b בננה***

***## 11 30 10 c גזר***

Alternatively, I could have pre-allocated a list with the names of the files.

df2\_list <- vector("list", length(files))

names(df2\_list) <- files

**for** (fname **in** files) {

df2\_list[[fname]] <- read.csv(fname)

}

2. מה קורה אילו השתמשתם ב- `for (nm in names(x))` ול-x אין שמות? מה קורה אילו רק לחלק מהרכיבים ב-x יש שמות? מה קורה אילו ישנם שמות שאינם ייחודיים (מצב בו אותו השם חוזר יותר מפעם אחת)?

*# a. vector with no names*

x <- c(11, 12, 13)

print(names(x))

***## NULL***

**for** (nm **in** names(x)) {

print(nm)

print(x[[nm]])

}

*# that's becaues:*

length(NULL)

***## [1] 0***

*# b. partial names - we get an error*

x <- c(a = 11, 12, c = 13)

names(x)

***## [1] "a" "" "c"***

**for** (nm **in** names(x)) {

print(nm)

print(x[[nm]])

}

***## [1] "a"***

***## [1] 11***

***## [1] ""***

***## Error in x[[nm]]: subscript out of bounds***

*# 3. duplicated names - only the first one is processed*

x <- c(a = 11, a = 12, c = 13)

names(x)

***## [1] "a" "a" "c"***

*#> [1] "a" "a" "c"*

**for** (nm **in** names(x)) {

print(nm)

print(x[[nm]])

}

***## [1] "a"***

***## [1] 11***

***## [1] "a"***

***## [1] 11***

***## [1] "c"***

***## [1] 13***

. כתוב פונקציה שמדפיסה את הממוצע של כל עמודה ב-`df` ויחד איתה גם את שם העמודה. למשל, `show\_mean(iris)` תדפיס:

**`show\_mean**(iris)

*# > Sepal.Length: 5.84*

*# > Sepal.Width: 3.06*

*# > Petal.Length: 3.76*

*# > Petal.Width: 1.20*`

show\_mean <- **function**(df) {

df2 <- df %>% select\_if(is.numeric)

output <- vector("numeric", ncol(df2))

names(output) <- names(df2)

**for** (i **in** seq\_along(df2)) {

output[[i]] <- mean(df2[[i]], na.rm = T)

}

output

}

**תרגיל 5.5 ופתרונותיו:**

1. חשב את הממוצע של כל עמודה ב mtcars.

2. קבע מהו סוג כל עמודה ב - `nycflights13::flights`

3. חשב כמה ערכים ייחודיים יש לכל עמודה ב-`iris`.

4. משוך 10 ערכים רנדומאליים מהתפלגות נורמאלית עם ממוצעים: 10-, 0, 10 ו-100.

**תשובות:**

1. To calculate the mean of every column in mtcars, apply the function mean() to each column, and use map\_dbl, since the results are numeric.

**map\_dbl**(mtcars, mean)

*#> mpg cyl disp hp drat wt qsec vs am gear*

*#> 20.091 6.188 230.722 146.688 3.597 3.217 17.849 0.438 0.406 3.688*

*#> carb*

*#> 2.812*

2. To calculate the type of every column in nycflights13::flights apply the function typeof(), discussed in the section on [Vector basics](https://r4ds.had.co.nz/vectors.html#vector-basics), and use map\_chr(), since the results are character.

**map\_chr**(nycflights13::flights, typeof)

*#> year month day dep\_time sched\_dep\_time*

*#> "integer" "integer" "integer" "integer" "integer"*

*#> dep\_delay arr\_time sched\_arr\_time arr\_delay carrier*

*#> "double" "integer" "integer" "double" "character"*

*#> flight tailnum origin dest air\_time*

*#> "integer" "character" "character" "character" "double"*

*#> distance hour minute time\_hour*

*#> "double" "double" "double" "double"*

3. The function n\_distinct() calculates the number of unique values in a vector.

**map\_int**(iris, n\_distinct)

*#> Sepal.Length Sepal.Width Petal.Length Petal.Width Species*

*#> 35 23 43 22 3*

The map\_int() function is used since length() returns an integer. However, the map\_dbl() function will also work.

**map\_dbl**(iris, n\_distinct)

An alternative to the n\_distinct() function is the expression, length(unique(...)). The n\_distinct() function is more concise and faster, but length(unique(...)) provides an example of using anonymous functions with map functions. An anonymous function can be written using the standard R syntax for a function:

**map\_int**(iris, **function**(x) **length**(**unique**(x)))

*#> Sepal.Length Sepal.Width Petal.Length Petal.Width Species*

*#> 35 23 43 22 3*

Additionally, map functions accept one-sided formulas as a more concise alternative to specify an anonymous function:

**map\_int**(iris, ~**length**(**unique**(.x)))

*#> Sepal.Length Sepal.Width Petal.Length Petal.Width Species*

*#> 35 23 43 22 3*

In this case, the anonymous function accepts one argument, which is referenced by .x in the expression length(unique(.x)).

4. To generate 10 random normals for each of μ= −10μ=−10,  0,  10, and 100: The result is a list of numeric vectors.

**map**(**c**(-10, 0, 10, 100), ~**rnorm**(n = 10, mean = .))

*#> [[1]]*

*#> [1] -9.56 -9.87 -10.83 -10.50 -11.19 -10.75 -8.54 -10.83 -9.71 -10.48*

*#>*

*#> [[2]]*

*#> [1] -0.6048 1.4601 0.1497 -1.4333 -0.0103 -0.2122 -0.9063 -2.1022 1.8934*

*#> [10] -0.9681*

*#>*

*#> [[3]]*

*#> [1] 9.90 10.24 10.06 7.82 9.88 10.11 10.01 11.88 12.16 10.71*

*#>*

*#> [[4]]*

*#> [1] 100.8 99.7 101.0 99.1 100.6 100.3 100.4 101.1 99.1 100*

Since a single call of rnorm() returns a numeric vector with a length greater than one we cannot use map\_dbl, which requires the function to return a numeric vector that is only length. The map functions pass any additional arguments to the function being called.

**תרגיל 8.7 ופתרונותיו:**

1. Create an enhanced col\_summary() that applies a summary function to every numeric column in a data frame

I will use map to apply the function to all the columns, and keep to only select numeric columns.

col\_sum2 <- **function**(df, f, ...) {

**map**(**keep**(df, is.numeric), f, ...)

}

**col\_sum2**(iris, mean)

*#> $Sepal.Length*

*#> [1] 5.84*

*#>*

*#> $Sepal.Width*

*#> [1] 3.06*

*#>*

*#> $Petal.Length*

*#> [1] 3.76*

*#>*

*#> $Petal.Width*

*#> [1] 1.2*

2. A possible base R equivalent of col\_summary() is:

col\_sum3 <- **function**(df, f) {

is\_num <- **sapply**(df, is.numeric)

df\_num <- df[, is\_num]

**sapply**(df\_num, f)

}

But it has a number of bugs as illustrated with the following inputs:

df <- **tibble**(

x = 1:3,

y = 3:1,

z = **c**("a", "b", "c")

)

*# OK*

**col\_sum3**(df, mean)

*# Has problems: don't always return numeric vector*

**col\_sum3**(df[1:2], mean)

**col\_sum3**(df[1], mean)

**col\_sum3**(df[0], mean)

What causes these bugs?

The cause of these bugs is the behavior of sapply(). The sapply() function does not guarantee the type of vector it returns, and will returns different types of vectors depending on its inputs. If no columns are selected, instead of returning an empty numeric vector, it returns an empty list. This causes an error since we can’t use a list with [.

**sapply**(df[0], is.numeric)

*#> named list()*

**sapply**(df[1], is.numeric)

*#> X1*

*#> TRUE*

**sapply**(df[1:2], is.numeric)

*#> X1 X2*

*#> TRUE FALSE*

The sapply() function tries to be helpful by simplifying the results, but this behavior can be counterproductive. It is okay to use the sapply() function interactively, but avoid programming with it.