This is my second blog post from the series of My R take on Advent of Code. If you’d like to know more about Advent of Code, check out the [first post from the series](https://r-tastic.co.uk/post/advent-of-code-in-r-day-1/) or simply go to [their website](https://adventofcode.com/2018). Below you’ll find the challnge from Day 2 and the solution that worked for me. As always, feel free to leave comments if you have different ideas on how this could have been solved!

**Day 2 Puzzle**

(…) you scan the likely candidate boxes again, counting the number that have an ID containing exactly two of any letter and then separately counting those with exactly three of any letter. You can multiply those two counts together to get a rudimentary checksum and compare it to what your device predicts.  
For example, if you see the following box IDs:

abcdef contains no letters that appear exactly two or three times.  
bababc contains two a and three b, so it counts for both.  
abbcde contains two b, but no letter appears exactly three times.  
abcccd contains three c, but no letter appears exactly two times.  
aabcdd contains two a and two d, but it only counts once.  
abcdee contains two e.  
ababab contains three a and three b, but it only counts once.

Of these box IDs, four of them contain a letter which appears exactly twice, and three of them contain a letter which appears exactly three times. Multiplying these together produces a checksum of 4 \* 3 = 12.  
What is the checksum for your list of box IDs?

So what is it all about? As complicated as it may sound, essentially we need to:

* understand which string contains letters that appear exactly 2 times
* understand which string contains letters that appear exactly 3 times
* count the number of each type of string
* multiply them together

Doesn’t sound so bad anymore, ey? This is how we can go about it:

First load your key packages…

library(dplyr)

library(stringr)

library(tibble)

library(purrr)

… and have a look at what the raw input looks like.

# check raw input

glimpse(input)

## chr "xrecqmdonskvzupalfkwhjctdb\nxrlgqmavnskvzupalfiwhjctdb\nxregqmyonskvzupalfiwhjpmdj\nareyqmyonskvzupalfiwhjcidb\"| \_\_truncated\_\_

Right, Advent of Code will never give you nice and clean data to work with, that’s for sure. But it doesn’t look like things are too bad this time – let’s just split it by the new line and keep it as a vector for now. Does it look reaosnably good?

# clean it

clean\_input = strsplit(input, '\n') %>% unlist() # splt by NewLine

glimpse(clean\_input)

## chr [1:250] "xrecqmdonskvzupalfkwhjctdb" "xrlgqmavnskvzupalfiwhjctdb" ...

Much better! Now, let’s put it all in a data frame for now, we’ll need it very soon.

# put it in the data.frame

df2 <- tibble(input = str\_trim(clean\_input))

head(df2)

## # A tibble: 6 x 1

## input

##

## 1 xrecqmdonskvzupalfkwhjctdb

## 2 xrlgqmavnskvzupalfiwhjctdb

## 3 xregqmyonskvzupalfiwhjpmdj

## 4 areyqmyonskvzupalfiwhjcidb

## 5 xregqpyonskvzuaalfiwhjctdy

## 6 xwegumyonskvzuphlfiwhjctdb

Now, the way I approached this was to split each word into letters and then count how many times they occured. Then, for identifying words with 2 occurences, I filtered only those that occur twice and if the final table has any rows, then this counts as yes. Take the first example:

strsplit(input, '\n') %>% unlist() %>% .[[1]] # get the first example

## [1] "xrecqmdonskvzupalfkwhjctdb"

Let’s split it by the letter, put it in a tibble and count each letter occurances:

strsplit(input, '\n') %>% unlist() %>% .[[1]] %>% # get the first example

strsplit('') %>% # split letters

unlist() %>% # get a vector

as\_tibble() %>% # trasform vector to tibble

rename\_(letters = names(.)[1]) %>% # name the column: letters

count(letters)

## # A tibble: 23 x 2

## letters n

##

## 1 a 1

## 2 b 1

## 3 c 2

## 4 d 2

## 5 e 1

## 6 f 1

## 7 h 1

## 8 j 1

## 9 k 2

## 10 l 1

## # ... with 13 more rows

Now, do we have any double occurances there?

# test: counting double letter occurances

strsplit(input, '\n') %>% unlist() %>% .[[1]] %>% # get the first example

strsplit('') %>% # split letters

unlist() %>% # get a vector

as\_tibble() %>% # trasform vector to tibble

rename\_(letters = names(.)[1]) %>% # name the column: letters

count(letters) %>% # count letter occurances

filter(n == 2) %>% # get only those with double occurances

nrow() # how many are there?

## [1] 3

Definitely yes. Let’s repeat the process for tripple occurances:

# test: counting triple letter occurances

strsplit(input, '\n') %>% unlist() %>% .[[1]] %>% # get the first example

strsplit('') %>% # split letters

unlist() %>%

as\_tibble() %>% # trasforming vector to tibble

rename\_(letters = names(.)[1]) %>%

count(letters) %>%

filter(n == 3) %>%

nrow()

## [1] 0

Not much luck with those in this case. To make our life easier, let’s wrap both calculations in functions…

### wrap-up in functions

# count double occurances

count2 <- function(x) {

result2 <- as.character(x) %>%

strsplit('') %>% # split by letters

unlist() %>%

as\_tibble() %>% # trasforming vector to tibble

rename\_(letters = names(.)[1]) %>%

count(letters) %>% # count letter occurances

filter(n == 2) %>%

nrow()

return(result2)

}

# count triple occurances

count3 <- function(x) {

result2 <- as.character(x) %>%

strsplit('') %>%

unlist() %>%

as\_tibble() %>% # trasforming vector to tibble

rename\_(letters = names(.)[1]) %>%

count(letters) %>%

filter(n == 3) %>%

nrow()

return(result2)

}

…and apply them to the whole dataset:

### apply functions to input

occurs2 <- map\_int(df2$input, count2)

occurs3 <- map\_int(df2$input, count3)

str(occurs2)

## int [1:250] 3 3 3 3 2 3 3 2 2 2 ...

Now, all we need to do is check how many positive elements we have in each vector and multiple their lengths by each other:

#solution

length(occurs2[occurs2 != 0]) \* length(occurs3[occurs3 != 0])

## [1] 5976

Voila!