```r
# Description -------------------------------------------------------------

# Sometimes, you have data in "fine resolution" and you need it actually
# at a more "aggregate" resolution. For instance, you may have montly
# data, but you actually need quarterly data (e.g. because you want
# to combine your data with other data where you only have quarterly
# information).

# In this script, we are having a look on how to do that.
# You can read more on this in the book in Chapter 3.



# Header ------------------------------------------------------------------


library(tidyverse)

rm(list = ls())

load("data/TradeEx_tidy.RData")



# Aggregating to quarterly data -------------------------------------------

# Calculate quarter


# We create a new dataframe Dq with quarterly data.
# We use the cut function that transforms a continuous
# range into a categorical variable
Dq <- D %>%
  mutate(quarter =  cut(month,c(0, 3.5, 6.5, 9.5, 13), labels = c("1", "2", "3", "4"))) %>%
  mutate(quarter = as.integer(quarter))

class(Dq$quarter)


# To see how the cut function works, consider

(checkItOut = cut(1:12,
    breaks = c(0, 3.5, 6.5, 9.5, 13),
    labels = c("1", "2", "3", "4")))

tibble(checkItOut)


# https://stackoverflow.com/questions/39041115/fixing-a-multiple-warning-unknown-column
#trade <- trade %>% select(-D0)
#trade$D0 = NA
```

```
# The next step is to "group" the data by quarters.
# Well, actually not by quarters, but by all possible
# combinations of years and quarters. (What's the difference?)
# For this, we use the group_by() function (see book in
# Chapter 3, Section "Grouped summaries with summarise()")

# The book only discusses summarise with hand-picked
# single columns. Here, we want to summarise a large
# number of columns. So we use the summarise_at()
# function that allows us to exactly do this.


# Get a list of all column names
names(Dq)

# We select columns 5 to 46 and send it to the
# vars() function that packages them as arguments
# for summarize_at().
# (Note, you can spell summarize with an s or z,
# R does not care, both functions do the same.)

x = vars(names(Dq)[5:46])


Dq = group_by(Dq,year, quarter) %>%
  summarize_at(x, mean)

# Note here some of the cool subtleties
# of R that make it a "functional" programming
# language. You can pass a function as an argument
# of a function: "mean" is a function that is an
# argument of summarize_at()!


# To show the power of functional programming,
# let's use another pair of functions.
# The code of the second line looks rather
# dense. I leave it to the experts among
# you to figure out how this works :-)
# For the others, it's good enough to
# understand what the code DOES, not how
# it works.

x = vars(names(Dq)[3:44])

Dq <- Dq %>% mutate_at(x, funs(round(., 2)))

# Check out that we now have indeed quarterly data,
# calculated as the means over 3 consecutive months!



# Changing order of columns ---------------------------------------------
```

```r
# Finally, let's explore how we can change the order of columns.

# Get the list of all column names
names(Dq)

# Selecting the columns that we want to have
# left-most, ordered appropriately
first = names(Dq)[c(1:2, 43:44)]

# Then using select() to rearange. Note the
# smart function everything(). It actually means
# "everything else"
Dq = select(Dq, first, everything())
```