QStep — Week 2 Lab: Homework Solutions

- Create a vector `even_numbers` of all even numbers from 0 to 100. Create a vector of their squares and call it `squares`. Create a vector of their square roots, and call it `roots`. Now bind the three vectors in a new dataframe with three columns.

```
even_numbers <- seq(0, 100, by = 2)
squares <- even_numbers^2
roots <- sqrt(even_numbers)

df <- data.frame(even_numbers, squares, roots)

head(df) # head() prints the first 6 rows of a dataframe
```

```
##   even_numbers squares    roots
## 1            0       0 0.000000
## 2            2       4 1.414214
## 3            4      16 2.000000
## 4            6      36 2.449490
## 5            8      64 2.828427
## 6           10     100 3.162278
```

- When we added `primes` and `one_to_ten`, we found that, as both vectors are of length 10, we got as a result another vector of length 10, where the first element is the sum of the first elements of the two original vectors, the second element is the sum of the second elements, and so on. What happens when we try to add vectors of different length? Go back to our vector `one_to_ten` (if needed, create it again), and let's try to add to it vectors of different length:

```
one_to_ten <- 1:10 #creates the vector
```

```
one_to_ten + 1000
```

```
##  [1] 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010
```

```
# adds 1000 to all elements
```

```
one_to_ten + c(1000,2000)
```

```
##  [1] 1001 2002 1003 2004 1005 2006 1007 2008 1009 2010
```

```
# adds 1000 to the first element, 2000 to the second, then 'recycles' the
# smaller vector adding 1000 to the third, 2000 to the fourth, and so on.
```

```
one_to_ten + c(1000,2000,3000)
```

```
## Warning in one_to_ten + c(1000, 2000, 3000): longer object length is not a
## multiple of shorter object length
```

```
##  [1] 1001 2002 3003 1004 2005 3006 1007 2008 3009 1010
```

```
# recycles the smaller vector, but returns a warning.
```

```
one_to_ten + c(1000,2000,3000,4000,5000)
```

```
##  [1] 1001 2002 3003 4004 5005 1006 2007 3008 4009 5010
```

```
# adds 1000 to the first element, 2000 to the second, 3000 to the third
# up until the fifth, then recycles
```

When doing operations with vectors of different length, the elements of the shorter vector are repeated so that it matches the larger vectors' length — i.e. the shorter vector is 'recycled' across the longer vector. If the longer vector is a multiple of the shorter vector — as when we add vectors of length 1,2 and 5 to `one_to_ten`, which is of length 10 — that's it. If the vectors' lengths are **not** multiples of each other, the shorter vector is partly recycled and the calculation is still performed as normal. However, R returns a warning, to indicate that this might be a mistake.

- Go back to the `brexit` dataset. Find out: (1) what's the mean percentage of Leave vote in a London constituency (variable `region`)?

```
brexit <- read.csv("brexit.csv") #loads the data

mean(brexit$percent_leave[brexit$region == "London"])
```

```
## [1] 39.09152
```

(2) what's the average Remain vote in areas classified as 'Predominantly Rural' (variable `area_type`; mind: there are missing values, so remember `na.rm = TRUE`)?

```
mean(brexit$percent_remain[brexit$area_type == "Predominantly Rural"], na.rm = TRUE)
```

```
## [1] 44.05429
```

(3) what's the total number of Leave votes (`leave_votes`) across all local authorities? What's the total number of Remain votes (`remain_votes`)?

```r
sum(brexit$leave_votes)
```

```
## [1] 17060477
```

```r
sum(brexit$remain_votes)
```

```
## [1] 15681212
```

```r
# Note: the totals are inexact, as we're missing Northern Ireland/Gibraltar
```

- Using indexing, find out what's the highest value of `median_age` in the dataset. Then find out which local authority corresponds to that value (i.e. what's the 'oldest' local authority in our dataset).

```r
# Solution 1
maximum_age <- max(brexit$median_age)
brexit$area[brexit$median_age == maximum_age]
```

```
## [1] "West Somerset"
```

```r
# Solution 2
brexit$area[brexit$median_age == max(brexit$median_age)]
```

```
## [1] "West Somerset"
```

```r
# Solution 3 (uses a new function)
brexit$area[which.max(brexit$median_age)]
```

```
## [1] "West Somerset"
```

- Create a new variable `result`, which takes four possible values: "Strong Leave" if Leave vote is 60% or over, "Weak Leave" if Leave vote is between 50 and 60%, "Weak Remain" if Leave vote is between 40 and 50%, and "Strong Remain" if Leave vote is under 40%. Remember to use the quote marks, as here you are working with character values. How many local authorities fall in each category?

```r
brexit$result <- NA
brexit$result[brexit$percent_leave >= 60] <- "Strong Leave"
brexit$result[brexit$percent_leave >= 50 & brexit$percent_leave < 60] <- "Weak Leave"
brexit$result[brexit$percent_leave >= 40 & brexit$percent_leave < 50] <- "Weak Remain"
brexit$result[brexit$percent_leave < 40] <- "Strong Remain"

table(brexit$result)
```

```
##
##  Strong Leave Strong Remain    Weak Leave   Weak Remain
##           102            42           161            75
```

What's the mean percentage of residents who identify as white (variable `percent_white`) in each of the four categories?

```
mean(brexit$percent_white[brexit$result == "Strong Leave"])
```

```
## [1] 93.30115
```

```
mean(brexit$percent_white[brexit$result == "Weak Leave"])
```

```
## [1] 92.27143
```

```
mean(brexit$percent_white[brexit$result == "Weak Remain"])
```

```
## [1] 87.88133
```

```
mean(brexit$percent_white[brexit$result == "Strong Remain"])
```

```
## [1] 79.81905
```

Pro tip: this can be done more elegantly using the more advanced `tapply` function. `tapply` takes the syntax `tapply(target_variable, group_variable, function)`. In this case, the target variable is `percent_white`, the group variable is `result`, and the function to apply is the `mean`.

```
tapply(brexit$percent_white, brexit$result, mean)
```

```
##  Strong Leave Strong Remain    Weak Leave   Weak Remain
##      93.30115      79.81905      92.27143      87.88133
```