Templating with Quarto

Bruno Rodrigues

10/28/22

# Set up

The goal is to have a frequency table for each question in a survey. But we do not want to have to do it by hand, so we define a function to create a table, and then, using the templating capabilities of Quarto, write some code to generate valid qmarkdown code. In the example below our survey only has 4 questions, but the solution described trivially scales to an infinity of questions. This is not the case if you’re solving this problem by hand.

Start by loading the data and defining some needed variables:

Let’s take a look at the data:

Table continues below

| Random.question. | Copy.of.Random.question. | Copy.of.Copy.of.Random.question. |
| --- | --- | --- |
| no | yes | yes |
| yes | yes | no |
| yes | no | no |
| yes | yes | yes |
| no | no | yes |
|  | yes | no |

| Copy.of.Copy.of.Copy.of.Random.question. |
| --- |
| yes |
| yes |
| no |
| yes |
| yes |
| no |

The column names are actually questions, so we save those in a variable:

[1] "Random.question."   
[2] "Copy.of.Random.question."   
[3] "Copy.of.Copy.of.Random.question."   
[4] "Copy.of.Copy.of.Copy.of.Random.question."

Now we define question codes:

[1] "var\_1" "var\_2" "var\_3" "var\_4"

We create a lookup table that links questions to their codes:

| codes | questions |
| --- | --- |
| var\_1 | Random.question. |
| var\_2 | Copy.of.Random.question. |
| var\_3 | Copy.of.Copy.of.Random.question. |
| var\_4 | Copy.of.Copy.of.Copy.of.Random.question. |

Finally, we replace the question names in the dataset by the code:

| var\_1 | var\_2 | var\_3 | var\_4 |
| --- | --- | --- | --- |
| no | yes | yes | yes |
| yes | yes | no | yes |
| yes | no | no | no |
| yes | yes | yes | yes |
| no | no | yes | yes |
|  | yes | no | no |
| no | yes | no | no |
|  |  | yes | yes |
| no | yes | no | no |
| no |  | yes | no |
| no | no | yes | yes |
| yes | yes | no | yes |
|  | no | no | yes |
| no | yes | no | no |
|  | no | no | no |
| yes | no | no |  |
|  | no | no | no |
| no | yes |  | yes |
| yes | no | no | yes |
| yes |  | no | yes |
| yes | no | yes | yes |
| no | no | yes | yes |
| no | yes | yes | yes |
|  | no |  | no |
| no | no | yes | no |
| yes | no | yes | yes |
| yes |  | yes | yes |
|  | no | no | yes |
| no | yes | yes | no |
| yes | no | yes | yes |
| yes | no | yes | yes |
| yes | no | no | yes |
| yes | yes | yes |  |
| yes | no | yes | no |
|  | no | no | no |
| yes | yes | no | no |
| yes | yes | no | no |
| no | yes |  | no |
|  | yes | yes | no |
| no | yes | no |  |
| yes |  | yes |  |
| yes | no |  | yes |
| no | no | no | no |
| yes | no | yes | yes |
| yes | no | no | no |
| no | yes | yes | no |
|  | yes | no | yes |
| no | no | no |  |
| yes | no | yes |  |
| yes | no | yes | no |
| yes | no | no | yes |
| no | no | no | yes |
| no | no | yes | no |
| yes |  | no | no |
| yes | no | yes | no |
| yes | yes | yes |  |
|  | yes | no | no |
| yes | no | yes | yes |
| yes | no | no | yes |
| no | no | yes | yes |
|  |  | yes |  |
| no |  | yes | no |
| no |  | yes |  |
| no | yes | no | yes |
| yes | no | no | yes |
| no |  | no | no |
| yes | yes | yes | no |
| yes | no | yes | no |
| no | yes | no | yes |
| no | no | yes | no |
| yes | no | no | yes |
|  | no | no | yes |
| no | yes | no | no |
| yes |  | yes | no |
| no | no | no | no |
| yes | yes | yes | yes |
| no | no | yes | yes |
| no | no |  | no |
| no | no | yes | no |
| no | yes | no | no |
|  | no |  | yes |
| no | no | yes | yes |
| yes | yes | no | yes |
| yes | no | yes | no |
| no | no | yes | no |
| yes |  | yes | no |
| no |  | no | yes |
| no | yes | yes | no |
| no | yes | yes | yes |
| yes | no |  | no |
|  | no | no | no |
|  |  | no | no |
| no | no | no | no |
| yes | yes | no | no |
| no | no | yes | yes |
| yes |  | yes |  |
| yes | yes | no | yes |
| yes |  | yes | no |
| yes | no | no | no |
| no | no | no | no |

Now, we define a function that creates a frequency table. This function has two arguments: dataset and var. It uses the dplyr::count() function to count each instance of the levels of var in dataset. Then it uses the knitr::kable() function. This functions takes a data frame as an argument and returns a table formatted in markdown code:

The next function is the one that does the magic: it takes only one argument as an input, and generates valid markdown code using the knitr::knit\_expand() function. Any variable between {{}} gets replaced by its value (so {{question}} gets replaced by the question that gets fetched from the lookup table defined above). Using this function, we can now loop over question codes, and what we get in return is valid markdown code that defines a section with the question as the title, and our table.

Our codes are strings, so to be able to use them inside of dplyr::count() we need to define them as bare string, or symbols. This can be done using the rlang::sym() function. If this is confusing, try running count(mtcars, "am") and you will see that it will not return what you want (compare to count(mtcars, am)). This is also why we needed rlang::quo\_name() in the function above, to convert the symbol back to a string, which is what grepl() requires:

Finally, we can create the sections. The line below uses purrr::walk(), which is equivalent to purrr::map(), the difference being that we use purrr::walk() when we are interested in the side effects of a function:

## Random.question.

| var\_1 | n |
| --- | --- |
|  | 16 |
| no | 40 |
| yes | 44 |

## Copy.of.Random.question.

| var\_2 | n |
| --- | --- |
|  | 16 |
| no | 52 |
| yes | 32 |

## Copy.of.Copy.of.Random.question.

| var\_3 | n |
| --- | --- |
|  | 7 |
| no | 46 |
| yes | 47 |

## Copy.of.Copy.of.Copy.of.Random.question.

| var\_4 | n |
| --- | --- |
|  | 10 |
| no | 48 |
| yes | 42 |