# Using RMarkdown and Quarto

Rich Cottrell

2023-05-02

## What are Rmarkdown/Quarto documents?

Rmarkdown (and the new generation Quarto documents) allow you to seamlessly integrate plain text and code into one document which helps to:

- Improve the description of what you are doing within each script
- Break's code and methods down into manageable chunks
- Documents how you approached your methods - so you can refer back to them in the future (e.g. for writing a manuscript).
- Write a manuscript or a report within R - every time you regenerate output files from Rmarkdown or Quarto documents, the figures and data products are updated to include any changes.
- Combine multiple coding languages if needed (R, Python, SQL, Bash)

### Stepping through some functions to manipulate data

To demonstrate how useful markdown documents can be for breaking down a method and describing your code we'll walk through some functions on to data that I use pretty regularly - global fisheries landings.

First let's load the libraries we want to use.

Now we can bring in the file using 'readRDS' and start tidying. The data is stored in RDS format rather than csv because it generates smaller files and that allows us to store it on github.

When you view the raw data you can see, the column names are horrrrrible - with spaces and parentheses - and the data is in wide format.

Seeing the output inline is one of the great aspects of R markdown. You can scroll through the data and check how the columns look super easily - this is a really good way of spotting errors.

```
(fisheries <- readRDS("data/input/fisheries_prod_raw.rds")) # reading in the compressed rds format data
```

```
## # A tibble: 22,347 x 72
##    `Country (Country)` `Species (ASFIS species)`    Fishing area (FAO major fi~1
##    <chr>               <chr>                        <chr>
##  1 Afghanistan         Freshwater fishes nei        Asia - Inland waters
##  2 Albania             Angelsharks, sand devils nei Mediterranean and Black Sea
##  3 Albania             Atlantic bluefin tuna        Mediterranean and Black Sea
##  4 Albania             Atlantic bonito              Mediterranean and Black Sea
##  5 Albania             Barracudas nei               Mediterranean and Black Sea
##  6 Albania             Bighead carp                 Europe - Inland waters
##  7 Albania             Bleak                        Europe - Inland waters
##  8 Albania             Blue and red shrimp          Mediterranean and Black Sea
##  9 Albania             Blue whiting(=Poutassou)     Mediterranean and Black Sea
## 10 Albania             Bluefish                     Mediterranean and Black Sea
## # i 22,337 more rows
## # i abbreviated name: 1: ‘Fishing area (FAO major fishing area)‘
## # i 69 more variables: ‘Unit (Unit)‘ <chr>, ‘1950‘ <chr>, ‘1951‘ <chr>,
## #   ‘1952‘ <chr>, ‘1953‘ <chr>, ‘1954‘ <chr>, ‘1955‘ <chr>, ‘1956‘ <chr>,
## #   ‘1957‘ <chr>, ‘1958‘ <chr>, ‘1959‘ <chr>, ‘1960‘ <chr>, ‘1961‘ <chr>,
## #   ‘1962‘ <chr>, ‘1963‘ <chr>, ‘1964‘ <chr>, ‘1965‘ <chr>, ‘1966‘ <chr>,
## #   ‘1967‘ <chr>, ‘1968‘ <chr>, ‘1969‘ <chr>, ‘1970‘ <chr>, ‘1971‘ <chr>, ...
```

Now we can start manipulating the data. I will use syntax from the tidyverse. This allows a continuous stream of code to perform consecutive functions on the same dataset, reducing code repetition. Here, I'm going to walk through each step and reproduce different versions of the data. In reality, I would integrate each of these into one pipeline, which I do at the end. But for now we can step through each one of these, breaking each move down, and illustrating the benefits of using markdown.

Using janitor to clean the names up is an easy first step, although not always what you want

```
(fisheries_clean <-
  fisheries |>
  clean_names()
)
```

```
## # A tibble: 22,347 x 72
##    country_country species_asfis_species fishing_area_fao_maj~1 unit_unit x1950
##    <chr>           <chr>                 <chr>                  <chr>     <chr>
##  1 Afghanistan     Freshwater fishes nei Asia - Inland waters   Tonnes -~ 100
##  2 Albania         Angelsharks, sand dev~ Mediterranean and Bla~ Tonnes -~ ...
##  3 Albania         Atlantic bluefin tuna Mediterranean and Bla~ Tonnes -~ -
##  4 Albania         Atlantic bonito       Mediterranean and Bla~ Tonnes -~ ...
##  5 Albania         Barracudas nei        Mediterranean and Bla~ Tonnes -~ ...
##  6 Albania         Bighead carp          Europe - Inland waters Tonnes -~ -
##  7 Albania         Bleak                 Europe - Inland waters Tonnes -~ ...
##  8 Albania         Blue and red shrimp   Mediterranean and Bla~ Tonnes -~ ...
##  9 Albania         Blue whiting(=Poutass~ Mediterranean and Bla~ Tonnes -~ ...
## 10 Albania         Bluefish              Mediterranean and Bla~ Tonnes -~ ...
## # i 22,337 more rows
## # i abbreviated name: 1: fishing_area_fao_major_fishing_area
## # i 67 more variables: x1951 <chr>, x1952 <chr>, x1953 <chr>, x1954 <chr>,
## #   x1955 <chr>, x1956 <chr>, x1957 <chr>, x1958 <chr>, x1959 <chr>,
## #   x1960 <chr>, x1961 <chr>, x1962 <chr>, x1963 <chr>, x1964 <chr>,
## #   x1965 <chr>, x1966 <chr>, x1967 <chr>, x1968 <chr>, x1969 <chr>,
## #   x1970 <chr>, x1971 <chr>, x1972 <chr>, x1973 <chr>, x1974 <chr>, ...
```

Next step is to convert the wide format to long. We can do this using pivot longer.

```
(fisheries_clean_1 <-
  fisheries_clean |>
  pivot_longer(names_to = "year", values_to = "value",  # so here we are saying we want the column name
               cols = -c(country_country, species_asfis_species,
                         fishing_area_fao_major_fishing_area, unit_unit)) #here we are saying this need
)
```

```
## # A tibble: 1,519,596 x 6
##    country_country species_asfis_species fishing_area_fao_majo~1 unit_unit year
##    <chr>           <chr>                 <chr>                   <chr>     <chr>
##  1 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes -- x1950
##  2 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes -- x1951
##  3 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes -- x1952
##  4 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes -- x1953
##  5 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes -- x1954
##  6 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes -- x1955
##  7 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes -- x1956
##  8 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes -- x1957
##  9 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes -- x1958
## 10 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes -- x1959
## # i 1,519,586 more rows
## # i abbreviated name: 1: fishing_area_fao_major_fishing_area
## # i 1 more variable: value <chr>
```

So the data is now in long format. But lets look at our data. What are the countries included for example, what are the classes of data stored? A quick check of the data summary shows the year and value data are still characters (not great for reasons we go into below). A quick look at unique countries included also shows that a few countries are based off totals or are the citation at the end of the spreadsheet. Lets remove them by using 'filter'

```
summary(fisheries_clean_1)

unique(fisheries_clean_1$country_country)

(fisheries_clean_2 <-
  fisheries_clean_1 |>
  filter(!country_country %in% c("Totals - Tonnes - live weight",
                                 "Totals - Number" ,
                                 "FAO. 2019. Fishery and Aquaculture Statistics. Global capture producti

)
```

Looking into the years and values as character variables. The reasons are the years now have an x before them which is not we want in any future figure or visualisation. If you scroll through the data, clicking the 'Next' button when in the viewer within markdown, you will see that the values (here supposed to be fish landings in tonnes) also have flags such as 'F' or are absent with only some symbols representing unknown amounts. In FAO fisheries data this is often:

"…" = data unavailable (here taken as zero) " " = data not separately available (here taken as zero) "-" = nil or zero (make zero) "0 0" = more than zero but less than half the unit used (taken to be 0.25) F = FAO estimate from available sources of information

The presence of these flags is often important information to retain but we don't want to keep it in the 'value column' (this makes it a character and you can't perform numerical functions on characters should we want to later). So we need to take the x's out of 'year' and the flags out of 'value', reporting the proper numerical values, and create a "flag" variable that retains that information. We can do this using 'mutate' which allows us to either mchange or mutate an existing column or create a new one that we specify.

```
(fisheries_clean_3 <-
  fisheries_clean_2 |>
  mutate(year = gsub("x", "", year) |>     #we can use mutate 'gsub' to edit the coumn 'year' by substit
            as.numeric(),                   #now make year numeric
          flag = case_when(value == "..." ~ "No data",                      # we can then use 'mutate'
                           value == " " ~ "Data not separately available",  # for the conditional proce
                           value == "-" ~ "Nil or zero",
                           value == "0 0" ~ "0<x<0.5",
                           grepl(" F", value) ~ "estimate",
                           TRUE ~ "Reported"),
          value = case_when(value %in% c("...", " ", "-") ~ "0",        # for all these flags, make val
                            value == "0 0" ~ "0.25",                    # for this flag make it halfway
                            grepl(" F", value) ~ gsub(" F", "", value), # where there is F - get rid of
                            TRUE ~ value) |>                           # for everything else keep it as
            as.numeric())                                               # make value numeric

)
```

```
## # A tibble: 1,519,392 x 7
##    country_country species_asfis_species fishing_area_fao_majo~1 unit_unit  year
##    <chr>           <chr>                 <chr>                   <chr>     <dbl>
##  1 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes --  1950
##  2 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes --  1951
##  3 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes --  1952
##  4 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes --  1953
##  5 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes --  1954
##  6 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes --  1955
##  7 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes --  1956
##  8 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes --  1957
##  9 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes --  1958
## 10 Afghanistan     Freshwater fishes nei Asia - Inland waters    Tonnes --  1959
## # i 1,519,382 more rows
## # i abbreviated name: 1: fishing_area_fao_major_fishing_area
## # i 2 more variables: value <dbl>, flag <chr>
```

This is looking much better. We can also choose to rename the variables using 'rename' if we are not happy with the way janitor did it. I typically do this further up to make use of more streamlined variable names for this whole process but we're just demonstrating. Also if you are comfortable with the data and the units that data are in (i.e. you work a lot with FAO fisheries data and know they report in tonnes), then you may also want to remove the unit column. We do this via 'select'

```
(fisheries_clean_4 <-
  fisheries_clean_3 |>
  rename(country = country_country,
         species = species_asfis_species,
         area = fishing_area_fao_major_fishing_area) |>
  select(-unit_unit)
)
```

```
## # A tibble: 1,519,392 x 6
##    country     species               area                  year value flag
##    <chr>       <chr>                 <chr>                <dbl> <dbl> <chr>
##  1 Afghanistan Freshwater fishes nei Asia - Inland waters  1950   100 Reported
##  2 Afghanistan Freshwater fishes nei Asia - Inland waters  1951   100 Reported
##  3 Afghanistan Freshwater fishes nei Asia - Inland waters  1952   100 Reported
##  4 Afghanistan Freshwater fishes nei Asia - Inland waters  1953   100 Reported
##  5 Afghanistan Freshwater fishes nei Asia - Inland waters  1954   100 Reported
##  6 Afghanistan Freshwater fishes nei Asia - Inland waters  1955   200 Reported
##  7 Afghanistan Freshwater fishes nei Asia - Inland waters  1956   200 Reported
##  8 Afghanistan Freshwater fishes nei Asia - Inland waters  1957   200 Reported
##  9 Afghanistan Freshwater fishes nei Asia - Inland waters  1958   200 Reported
## 10 Afghanistan Freshwater fishes nei Asia - Inland waters  1959   200 Reported
## # i 1,519,382 more rows
```

Now the data is the way you want it to be, in terms of variable class, nomenclature, and long format we can start manipulating. Say you want to know how much a country produces each year - but you don't care about the species or the area (inland or marine) - we can group by the variables we are interested in (country and year) using 'group_by' and summarise that data into a new variable using 'summarise'. Don't forget to ungroup()!

```r
(fisheries_clean_5 <-
  fisheries_clean_4 |>
  group_by(country, year) |>
  summarise(sum_value = sum(value, na.rm = TRUE)) |>
  ungroup()
)
```

```
## # A tibble: 16,796 x 3
##    country      year sum_value
##    <chr>       <dbl>     <dbl>
##  1 Afghanistan  1950       100
##  2 Afghanistan  1951       100
##  3 Afghanistan  1952       100
##  4 Afghanistan  1953       100
##  5 Afghanistan  1954       100
##  6 Afghanistan  1955       200
##  7 Afghanistan  1956       200
##  8 Afghanistan  1957       200
##  9 Afghanistan  1958       200
## 10 Afghanistan  1959       200
## # i 16,786 more rows
```

We can then add other columns that are a function of existing ones. Sometimes this means you need to operate on each set of values independently as if they were different data frames. A combination of group_by() and nest() splits your data frame into a list of data frames based on your grouping variables. Then using mutate we can map a function onto your split dataframes using map from the purrr package to create the new column. You can then unnest() and ungroup() to bring it all together again at the end. I'll demonstrate by getting the relative production for each country (relative to 1950 that is).

```r
(fisheries_clean_6 <-
  fisheries_clean_5 |>
  group_by(country) |>
  nest() |>
  mutate(rel_value = purrr::map(data, ~(.$sum_value/.$sum_value[1]))) |>
  unnest(cols = c(data, rel_value)) |>
  ungroup()
  )
```

```
## # A tibble: 16,796 x 4
##    country      year sum_value rel_value
##    <chr>       <dbl>     <dbl>     <dbl>
##  1 Afghanistan  1950       100         1
##  2 Afghanistan  1951       100         1
##  3 Afghanistan  1952       100         1
##  4 Afghanistan  1953       100         1
##  5 Afghanistan  1954       100         1
##  6 Afghanistan  1955       200         2
##  7 Afghanistan  1956       200         2
##  8 Afghanistan  1957       200         2
##  9 Afghanistan  1958       200         2
## 10 Afghanistan  1959       200         2
## # i 16,786 more rows
```

Another really useful set of functions are the join functions. IF you had a dataframe where you wanted to add additional information for each country and year for example, a left_join() could be really useful. This join type is a left join because when joing data frames/ tibbles A and B (in that order), it will match the datasets using variables you dictate. All of the information of data frame A is retained in the left join and matched to those relevant in data frame B.

right_join() retains all relevant information in df B, inner_join() retains only information both data sets possess, full_join() retains all data from both dataframes. You need to pick the one right for your needs. Here I demonstrate a left_join with a dataset with fake GDP data for Afghanistan.

```
GDP <- tibble(country = "Afghanistan", year = c(1950, 1951), gdp_millions = c(3000, 4000))

(fisheries_clean_7 <-
  fisheries_clean_6 |>
  left_join(GDP, by = c("country", "year")) #notice the rest of the GDP variable is returned as NA wher
)
```

```
## # A tibble: 16,796 x 5
##    country      year sum_value rel_value gdp_millions
##    <chr>       <dbl>     <dbl>     <dbl>        <dbl>
##  1 Afghanistan  1950       100         1         3000
##  2 Afghanistan  1951       100         1         4000
##  3 Afghanistan  1952       100         1           NA
##  4 Afghanistan  1953       100         1           NA
##  5 Afghanistan  1954       100         1           NA
##  6 Afghanistan  1955       200         2           NA
##  7 Afghanistan  1956       200         2           NA
##  8 Afghanistan  1957       200         2           NA
##  9 Afghanistan  1958       200         2           NA
## 10 Afghanistan  1959       200         2           NA
## # i 16,786 more rows
```

Usually you wouldn't need to create this many objects but it's easy to show this way. Below is the whole process redone in one continuous pipe. Once done you are ready to start playing around with your data, adding more variables, and presenting it. Happy wrangling!

```
(fisheries_final <-
    readRDS("data/input/fisheries_prod_raw.rds") |>
    clean_names() |>
    pivot_longer(names_to = "year", values_to = "value",
                 cols = -c(country_country, species_asfis_species, fishing_area_fao_major_fishing_area,
                   filter(!country_country %in% c("Totals - Tonnes - live weight",   "Totals - Number"
                   mutate(year = gsub("x", "", year) |>
              as.numeric(),
            flag = case_when(value == "..." ~ "No data",
                             value == " " ~ "Data not separately available",
                             value == "-" ~ "Nil or zero",
                             value == "0 0" ~ "0<x<0.5",
                             grepl(" F", value) ~ "estimate",
                             TRUE ~ "Reported"),
            value = case_when(value %in% c("...", " ", "-") ~ "0",
                              value == "0 0" ~ "0.25",
                              grepl(" F", value) ~ gsub(" F", "", value),
                              TRUE ~ value) |>
              as.numeric()) |>
    rename(country = country_country,
           species = species_asfis_species,
           area = fishing_area_fao_major_fishing_area) |>
    select(-unit_unit) |>
    group_by(country, year) |>
    summarise(sum_value = sum(value, na.rm = TRUE)) |>
    ungroup() |>
    group_by(country) |>
    nest() |>
  mutate(rel_value = purrr::map(data, ~(.$sum_value/.$sum_value[1]))) |>
  unnest(cols = c(data, rel_value)) |>
  ungroup() |>
  left_join(GDP, by = c("country", "year"))
                 )
```

```
## # A tibble: 16,796 x 5
##    country      year sum_value rel_value gdp_millions
##    <chr>       <dbl>     <dbl>     <dbl>        <dbl>
##  1 Afghanistan  1950       100         1         3000
##  2 Afghanistan  1951       100         1         4000
##  3 Afghanistan  1952       100         1           NA
##  4 Afghanistan  1953       100         1           NA
##  5 Afghanistan  1954       100         1           NA
##  6 Afghanistan  1955       200         2           NA
##  7 Afghanistan  1956       200         2           NA
##  8 Afghanistan  1957       200         2           NA
##  9 Afghanistan  1958       200         2           NA
## 10 Afghanistan  1959       200         2           NA
## # i 16,786 more rows
```

Note there are numerous options you can set in the code chunks. In the first setup chunk with libraries I used the term 'include = FALSE',. This tells R to run the chunk when knitting but do not include the code or the code output in the knitted document.

Options cover: - 'include = FALSE' - runs the code but does not print the output or include the code in the knitted document - 'echo = FALSE' - which runs the code and prints the output but does not include the code in the knitted document - 'message = FALSE' - prevents messages generated by code appearing in knitted document - ‘warning = FALSE’ - prevents warnings generated by code appearing in knitted document - ‘eval = FALSE’ - when combined with echo = FALSE this tells R to completely disregard this chunk during knitting. - ‘results = ’hide’ ’ - this tells R to include the code but hide the output

Note I am running bash syntax in this chunk - for other languages you with need more setup

```
ls -al ~/.ssh
```

```
## total 24
## drwx------ 2 richardcottrell users 4096 Apr 23  2022 .
## drwxr-xr-x 9 richardcottrell users 4096 Apr 14 04:54 ..
## -rw------- 1 richardcottrell users  399 Apr 22  2022 authorized_keys
## -rw------- 1 richardcottrell users 2622 Apr 23  2022 id_rsa
## -rw-r--r-- 1 richardcottrell users  581 Apr 23  2022 id_rsa.pub
## -rw-r--r-- 1 richardcottrell users 1110 Jun 29  2022 known_hosts
```

Here I tell R to ignore this chunk - and so nothing prints.

Here I tell R to run the code and include it in my knitted document but hide the output

```
ls -al ~/.ssh
```