Data cleaning procedure in R

Thao Tran

Last edited 16 January 2023

Purpose: Importance of transparency and reproducibility

The replication crisis calls attention to the credibility of many psychological research (see Ioannidis (2005), Ioannidis et al. (2012), Nosek et al. (2015) for some examples). In a recent review paper, Hardwicke et al. (2021) noted that transparency and reproducibility-related practices are still lacking. Making research materials available and providing step-by-step description of how the researchers arrive at a particular conclusion are critically important. This includes talking about data cleaning and any decisions researchers make during the process of making their data ready for further analysis.

Cleaning up data with a programming language like R and exporting a document including every step of the data cleaning and wrangling process have several benefits. First, such document serves as a contract between the researchers and the consumers of research, noting what have been done with complete transparency. Second, this truly makes the research process time-proof. If the researchers need to revisit the data and the analysis for any reason after a particular project is concluded, they will have no difficulty re-evaluate their past work. Further, the research results will also be perfectly reproducible for future researchers. Third, some researchers may find it useful to do some sensitivity analysis with their data and see if their results will change if they make different the data cleaning/wrangling decisions.

In short, cleaning up data in a programming language like R is a big time investment up front, but it will save researchers considerable efforts down the road. For such reason, this data cleaning codebook is written to walk researchers through real practical example of how one can manipulate data in R, from reading the dataset downloaded from a survey collection platform like Qualtrics, to cleaning up the variables' format, fixing up encoding errors during the data collection process, and exporting the cleaned data to a file for analysis and dissemination.

Data introduction

In this demonstration codebook, I used the COVIDiSTRESS data on individuals' behaviors and attitudes amidst the COVID-19 pandemic. The data was collected from March 2020 to May of the same year. Administered in multiple languages, this global survey data is complex enough for a realistic demonstration of the data preparation in R. A detailed report on the survey can be found in Yamada et al. (2021), but readers can find a short description of the survey variables below:

- **Dem_**: Variables with this prefix include background information about the participants, such as their age, gender, level of education, employment status, country of residence, expat status, state/province that they live in, marital status, number of dependents, etc.
- **AD_gain** and **AD_loss**: These 2 variables describe participants' choices in the Asian Disease Scenario (Tversky & Kahneman, 1981)
- **AD_check**: This question checks if participants have previously seen the two `AD_X` questions before taking part in the COVIDiSRESS global survey.

- **Scale_PSS10_UCLA_**: There are 13 variables with this prefix in the raw data file. The first ten questions ask for participants' perceived stress (Cohen et al., 1983) while the last three gauge for perceived loneliness (Hughes et al., 2004) over the past week.
- **OCED_people_**: There are 2 variables with this prefix in the raw data file measuring participants' level of trust in (1) most people, and (2) most people they know personally.
- **OCED_institutions_**: There are 6 variables with this prefix in the raw data file measuring participants' level of trust in different governmental institutions.
- **Corona_concerns_**: There are 5 variables with this prefix in the raw data file measuring participants' level of concern about consequences of coronavirus for themselves and others.
- **Trust_countrymeasure**: This variable asks if participants' trust in the appropriateness of their government's responses to coronavirus.
- **Compliance_**: There are 6 variables with this prefix in the raw data file measuring participants' awareness and compliance with public health guidelines to combat the COVID-19.
- **BFF_15_**: There are 15 variables with this prefix measuring different aspects of participants' personality (Lang et al., 2011)
- **Expl_Distress_**: There are 15 variables with this prefix in the raw data file measuring participants' agreement on different sources of psychological distress during the coronavirus pandemic (e.g., day-to-day income, children's education, future jobs, etc.).
- **Expl_Distress_txt**: This free-text variable asks participants to write about any other sources of distress they may have.
- **SPS_**: There are 10 variables with this prefix in the raw data file measuring participants' ratings on the availability of social provisions during distressing situations (Steigen & Bergh, 2019).
- **Expl_Coping_**: There are 16 variables with this prefix in the raw data file measuring participants' agreement on different strategies to reduce psychological distress.
- **Expl_coping_txt**: This free-text variable asks participants to write about how they cope with psychological distress.
- **Expl_media_**: There are 6 variables with this prefix in the raw data file measuring participants' agreement on the use of different media sources for coronavirus-related information.
- **Final_open**: This final free-text variable serves as a space for participants to share any remaining thoughts they would like to share with the research team.

Set up: load packages and dataset

This data cleaning document was created fully in R Markdown. To start, I read in some useful R packages that help me read the data in R and allow me to manipulate the imported data file efficiently.

Necessary packages

```r
#install.packages("qualtRics")
#install.packages("dplyr")
#install.packages("stringr")
#install.packages("multicon")

library(qualtRics)
library(dplyr)
library(stringr)
library(multicon)
```

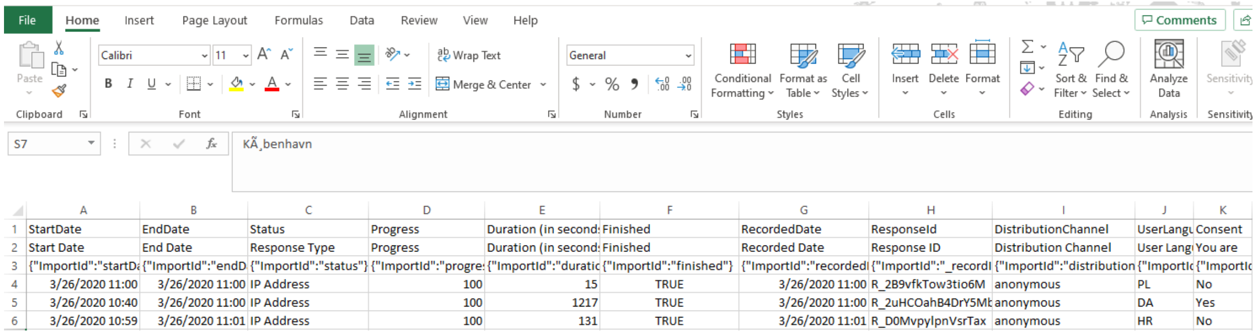
To use a R package, I needed to install it on my device, and load it in my working environment:

- The `install.packages()` function: As its name suggests, this function looks for the specified package (e.g., tidyverse) and install it on my device. By default, this function searches for packages located in the CRAN repository (an official network maintained by the global R community).
  - I already installed the specified packages. Thus, I turned the function into a comment/note to myself by adding a hashtag (#) in front of the function. *Note: For any package, I only need to install it once. Then, I can load it in all subsequent R projects that I work on.*
  - If I want to install packages from repositories other than the CRAN repository, I can follow the instructions here.
- The `library()` function: This function loads packages in my working environment and allows me to use the functionalities of the installed and loaded packages.
  - `packagename::functionname()`: In some cases, I am only interested in a specific function within a package, I can use this notation instead of loading the whole package with `library()`.

Load dataset

```
#This read_survey function from qualtRics keep the header and remove the first two non-data rows
MyData <- qualtRics::read_survey(file_name = "input/COVIDiSTRESS-global-survey-May302020.csv")
```

I downloaded the data from Qualtrics to my local device, and saved it in a folder called **input** that is a sub-folder within my working directory. Social scientists who frequently work with Qualtrics may realize that the file downloaded from Qualtrics often include repetitive introductions to matrix questions (as demonstrated below).



S7 KÃ¸benhavn

	A	B	C	D	E	F	G	H	I	J	K
1	StartDate	EndDate	Status	Progress	Duration (in second:	Finished	RecordedDate	ResponseId	DistributionChannel	UserLangu	Consent
2	Start Date	End Date	Response Type	Progress	Duration (in second:	Finished	Recorded Date	Response ID	Distribution Channel	User Lang	You are
3	{"ImportId":"startDa	{"ImportId":"endD	{"ImportId":"status"}	{"ImportId":"progre:	{"ImportId":"duratic	{"ImportId":"finished"}	{"ImportId":"recordedI	{"ImportId":"_recordI	{"ImportId":"distribution	{"ImportIc	{"ImportIc
4	3/26/2020 11:00	3/26/2020 11:00	IP Address	100	15	TRUE	3/26/2020 11:00	R_2B9vfkTow3tio6M	anonymous	PL	No
5	3/26/2020 10:40	3/26/2020 11:00	IP Address	100	1217	TRUE	3/26/2020 11:00	R_2uHCOahB4DrY5Mb	anonymous	DA	Yes
6	3/26/2020 10:59	3/26/2020 11:01	IP Address	100	131	TRUE	3/26/2020 11:01	R_D0MvpylpnVsrTax	anonymous	HR	No

- I used the `read_survey()` function from the qualtRics package to read comma separated CSV files generated by Qualtrics into R:
  - There are many arguments in this function, but for simplicity, we are only going to use the first argument **file_name**.
  - To read in your own data file, you should replace **input/COVIDiSTRESS-global-survey-May302020.csv** with the relative path to your data file.

Data cleaning procedures

Examine the data structure

Before proceeding to clean the data, I needed to make sure that the data is read in properly in R. In other words, I wanted to ensure that R correctly recognizes the types of variables I have in the data (e.g. numbers are correctly recognized as numeric). I did so with the `str()` function. The function printed out every variable I have in the data, together with their types and example values.

Note: The printout is rather long, so I only printed out the structure of the first six columns.

```
#str(MyData)

str(MyData[,1:6])
```

```
## tibble [173,426 x 6] (S3: tbl_df/tbl/data.frame)
##  $ StartDate            : POSIXct[1:173426], format: "2020-03-26 11:00:17" "2020-03-26 10:40:16" ...
##  $ EndDate              : POSIXct[1:173426], format: "2020-03-26 11:00:32" "2020-03-26 11:00:33" ...
##  $ Status               : chr [1:173426] "IP Address" "IP Address" "IP Address" "IP Address" ...
##   ..- attr(*, "label")= Named chr "Response Type"
##   .. ..- attr(*, "names")= chr "Status"
##  $ Progress             : num [1:173426] 100 100 100 100 100 100 100 100 100 100 ...
##   ..- attr(*, "label")= Named chr "Progress"
##   .. ..- attr(*, "names")= chr "Progress"
##  $ Duration (in seconds): num [1:173426] 15 1217 131 770 15 ...
##   ..- attr(*, "label")= Named chr "Duration (in seconds)"
##   .. ..- attr(*, "names")= chr "Duration (in seconds)"
##  $ Finished             : logi [1:173426] TRUE TRUE TRUE TRUE TRUE TRUE ...
##   ..- attr(*, "label")= Named chr "Finished"
##   .. ..- attr(*, "names")= chr "Finished"
##  - attr(*, "column_map")= tibble [151 x 7] (S3: tbl_df/tbl/data.frame)
##   ..$ qname      : chr [1:151] "StartDate" "EndDate" "Status" "Progress" ...
##   ..$ description: chr [1:151] "Start Date" "End Date" "Response Type" "Progress" ...
##   ..$ main       : chr [1:151] "Start Date" "End Date" "Response Type" "Progress" ...
##   ..$ sub        : chr [1:151] "" "" "" "" ...
##   ..$ ImportId   : chr [1:151] "startDate" "endDate" "status" "progress" ...
##   ..$ timeZone   : chr [1:151] "America/Denver" "America/Denver" NA NA ...
##   ..$ choiceId   : logi [1:151] NA NA NA NA NA NA ...
```

Filter for cases with consent and under 18 years of age

The research team received a waiver from the IRB office at Aarhus University, Denmark to collect responses to this survey. The target population of the survey was adults who gave consent to be part of the data collection process. As such, respondents who did not give consent or people who indicated that they were below the age of 18 were excluded from the final data.

```
# Filter for cases with consent and participants below 18
# Save as a new running file to see how many participants left out after each step
MyFilteredData <- MyData %>%
  filter(Consent == "Yes" & Dem_age >= 18)
```

In order to see the number of participants discarded after each cleaning step, I did not overwrite the original data. Instead, I saved a new object called `MyFilteredData`. Here is the breakdown of the code above:

- Took `MyData` object.
- Used the `filter()` function from the dplyr package to subset `MyData` when certain conditions are met:
  - Here, when the variable `Consent` was equal to **Yes** AND when the variable `Dem_age` variable was equal or greater than **18**, I told R to keep the responses.

- All retained responses were saved in an object called `MyFilteredData`

The original sample size was 173426. After filtering out cases under 18 and did not give consent, I discarded 47451 and ended up with a sample size of 125975.

Filter for official launch date of the survey

After testing out the survey, the COVIDiSTRESS team decided that the official launch date for the survey would be March 30 of 2020 for participating countries. Two exceptions were Denmark and Kosovo where the survey was opened earlier. As a result, records from countries other than Denmark and Kosovo before March 30 of 2020 needed to be removed.

Set timezone when working with date-time object

The COVIDiSTRESS survey was administered in different countries in multiple time zones. However, Qualtrics recorded time in the Coordinated Universal Time (UTC). To begin working with date-time information, I needed to confirm if R correctly recognizes the survey completion date column (EndDate) as a date-time data type. Then, I also changed the time zone on my local device as **UTC**.

```
#Check the structure of EndDate, which will be used to filter responses
str(MyFilteredData$EndDate)
```

```
##  POSIXct[1:125975], format: "2020-03-26 11:00:33" "2020-03-26 11:03:12" "2020-03-26 11:19:17" ...
```

```
#Change R time zone to UTC
Sys.setenv(TZ='UTC')
```

Good! R correctly identified **EndDate** as a `POSIXct-class` variable. Storing date-time information as a `POSIXct` class optimizes uses (allowing for subsetting, ordering, calculating time differences, etc.), speeds up computation, processing, and conversion to other formats.

Here is an explanation of the codes above: - I called the `str()` function here again, but I referred to a single column in my data object with the `$` sign. - The `Sys.setenv()` allows me to set the timezone in my local device to UTC.

Recode misaligned responses

Before filtering out responses before the official launch date, I received notes from the collaborators that there were some issues with the master sheet we uploaded to Qualtrics for multi-language survey. The errors in the master sheet caused misalignment of the response options for the country of residence question for the Bulgarian and Afrikaans version of the survey. I needed to fix this issue before I could filter out data that came in before the official launch date.

Note: This is an unlikely, but not entirely impossible issue that researchers may need to address when working with cross-cultural studies using multiple survey languages. I chose not to go further into how this issue emerged but rather focus on how to address something like this when it happens.

Recode responses for the Bulgarian language

Before recoding the responses, I had checked in with the people that managed the Qualtric master sheet for the timestamp when the error was addressed. They confirmed that error was fixed on **2020-04-08** at **07:48:00**. The code below showed the number of people who responded to the survey using the Bulgarian language and where they were from. Here, I used the `filter()` function from the dplyr package again. I told R to look for participants who answered the question in Bulgarian (`UserLanguage` is equal to **BG**) and who started the survey before or at a certain time (`StartDate` is greater than/equal to **2020-04-08 7:48:00**). Then, I counted the total number of observations who met these conditions to get a sense of what we had to deal with.

```
MyFilteredData %>%
  filter(UserLanguage == "BG", StartDate <= "2020-04-08 07:48:00") %>%
  count(Country)
```

```
## # A tibble: 45 x 2
##    Country                 n
##    <chr>               <int>
##  1 - other                14
##  2 Afghanistan             1
##  3 Algeria                 1
##  4 Angola                  1
##  5 Antigua and Barbuda     1
##  6 Armenia                 2
##  7 Australia               8
##  8 Belarus                11
##  9 Brunei               3849
## 10 Bulgaria               93
## # ... with 35 more rows
```

Using the information obtained from people who managed the different survey languages, I was able to recode the country names properly.

```
#Move 1 country down for people responding in BG
MyFilteredData <- MyFilteredData %>%
  mutate(Country = ifelse(
    StartDate <= "2020-04-08 07:48:00" & UserLanguage == "BG",
    case_when(Country == "- other" ~ NA_character_,
      Country == "Afghanistan" ~ "Afghanistan",
      Country == "Algeria" ~ "Andorra",
      Country == "Angola" ~ "Antigua and Barbuda",
      Country == "Antigua and Barbuda" ~ "Argentina",
      Country == "Armenia" ~ "Australia",
      Country == "Australia" ~ "Austria",
      Country == "Belarus" ~ "Belgium",
      Country == "Brunei" ~ "Bulgaria",
      Country == "Bulgaria" ~ "Burkina Faso",
      Country == "Cameroon" ~ "Canada",
      Country == "Cuba" ~ "Cyprus",
      Country == "Cyprus" ~ "Czech Republic",
      Country == "Czech Republic" ~ "Denmark",
      Country == "Fiji" ~ "Finland",
      Country == "Finland" ~ "France",
      Country == "Georgia" ~ "Germany",
      Country == "Ghana" ~ "Greece",
      Country == "Iraq" ~ "Ireland",
      Country == "Ireland" ~ "Israel",
      Country == "Israel" ~ "Italy",
      Country == "Japan" ~ "Jordan",
      Country == "Korea, North" ~ "Korea, South",
      Country == "Korea, South" ~ "Kosovo",
      Country == "Liechtenstein" ~ "Lithuania",
      Country == "Lithuania" ~ "Luxembourg",
      Country == "Mali" ~ "Malta",
      Country == "Nepal" ~ "Netherlands",
      Country == "Nigeria" ~ "North Macedonia",
      Country == "Poland" ~ "Portugal",
      Country == "Portugal" ~ "Qatar",
      Country == "Qatar" ~ "Romania",
      Country == "South Africa" ~ "Spain",
      Country == "Spain" ~ "Sri Lanka",
      Country == "Suriname" ~ "Sweden",
      Country == "Sweden" ~ "Switzerland",
      Country == "Tanzania" ~ "Thailand",
      Country == "The Bahamas" ~ "Bahrain",
      Country == "Tunisia" ~ "Turkey",
      Country == "Uganda" ~ "Ukraine",
      Country == "Ukraine" ~ "United Arab Emirates",
      Country == "United Arab Emirates" ~ "United Kingdom",
      Country == "United Kingdom" ~ "United States",
      Country == "Zimbabwe" ~ NA_character_),
   Country))
```

In the code chunk above, I took the following steps:

- I used the `mutate()` function (from the dplyr package) to create a variable in `MyFilteredData` dataframe. Since I used an existing variable in my dataset - `Country`, I essentially overwrote it.
- Inside the `mutate()` function, I used the `ifelse()` function to assign value to the variable `Country`:
  - The first argument of the `ifelse()` function is a logical test. The second argument contains the values of `Country` when the logical test is **TRUE**. The third argument contains the values of `Country` if such test is **FALSE**.
    - Here, I wanted to see if the `StartDate` is before or at the time when the error happened [and].{underline} if the language used to answer the survey is Bulgarian (`UserLanguage` is equal to **BG**).
    - If the logical test is **TRUE**, I used the `case_when()` function (also from the dplyr package) to overwrite the `Country` variable. For example, if the original value in `Country` is "Algeria", overwrite it "Andorra".
    - If the logical test is **FALSE**, I told R to keep the original values in the `Country` variable.

Recode responses for the Afrikaans language

I did the same thing for this version of the survey, using the `filter()` function to check for misaligned cases.

```
#Before we recode country on Qualtrics for the BG language, how many people answer the survey in BG, an
MyFilteredData %>%
  filter(UserLanguage == "AFR", StartDate <= "2020-04-07 06:48:00") %>%
  count(Country)
```

```
## # A tibble: 2 x 2
##   Country            n
##   <chr>          <int>
## 1 Somalia            4
## 2 United Kingdom     1
```

Since there were only a couple cases to fix, I used two nested `ifelse()` functions to recode these. I told R to look for cases that responded at or before **2020-04-07 06:48:00** and used the **AFR** language code to answer the survey. Among cases that met this condition, I told R to evaluate if the original value in `Country` is “Somalia” and assign it with a new value - “South Africa”. If the original value in `Country` is not “Somalia”, assign it with a different new value - “United States”. Then, I closed the inner `ifelse()` function, and specified R to keep all original values in `Country` if `UserLanguage` is not **AFR** and `StartDate` is after the specified time.

```
MyFilteredData <- MyFilteredData %>%
  mutate(Country = ifelse(
    UserLanguage == "AFR" & StartDate <= "2020-04-07 06:48:00",
      ifelse(Country == "Somalia", "South Africa", "United States"), Country))
```

Filter for cases before the official launch date

The official launch date for the survey was March 30 of 2020 for most participating countries, excepting Denmark and Kosovo. Hence, I needed to remove records from participants outside of Denmark and Kosovo before March 30 of 2021.

```
BeforeLaunch <- MyFilteredData %>%
  filter(!Country %in% c("Denmark", "Kosovo") &  StartDate <= "2020-03-30 00:00:00")

MyFilteredData <- MyFilteredData %>%
  anti_join(BeforeLaunch)
```

I took a slightly different approach here to see how much the sample size reduces after removing pre-launch cases.

- I used the `filter()` function to create a subset of the data called `BeforeLaunch` containing cases that are (1) outside of Denmark and Kosovo and (2) recorded before 2020-03-30. The exclamation point **!** negates/reverses the first condition.
- I used the `anti_join()` function (from the dplyr package) to remove cases within the `BeforeLaunch` subset from the main `MyFilteredData` dataset.

After removing pre-launch cases, my sample size decreased from 125975 to 125306.

Rearrange marital status for some languages

Similar to the problem with country of residence, the question on marital status was not positioned correctly, resulting in misalignment between the English and non-English survey versions. Here, I used the `mutate()` function, together with the `ifelse()` and `case_when()` functions again.

```
MyFilteredData <- MyFilteredData %>%
  mutate(Dem_maritalstatus =
           ifelse(UserLanguage != "EN",
                  case_when(
                    Dem_maritalstatus == "Single" ~ "Other or would rather not say",
                    Dem_maritalstatus == "Married/cohabiting" ~ "Single",
                    Dem_maritalstatus == "Divorced/widowed" ~ "Married/cohabiting",
                    Dem_maritalstatus == "Other or would rather not say" ~ "Divorced/widowed",
                    TRUE ~ Dem_maritalstatus),
                  Dem_maritalstatus))
```

Recode uninformative responses

When examining the demographic variables in the survey, the team found out that there were some uninformative responses in the marital status (`Dem_maritalstatus`) and educational level questions (`Dem_edu` and `Dem_edu_mom`).

```
unique(MyFilteredData$Dem_maritalstatus)
```

```
## [1] "Single"                        "Married/cohabiting"
## [3] "Divorced/widowed"              "5"
## [5] "Other or would rather not say" NA
```

For example, some people indicated that their marital status is “5”.

```
unique(MyFilteredData$Dem_edu)
```

```
## [1] "- Up to 12 years of school"
## [2] "- Some College, short continuing education or equivalent"
## [3] "- College degree, bachelor, master"
## [4] "- Up to 9 years of school"
## [5] "- Up to 6 years of school"
## [6] "- PhD/Doctorate"
## [7] "- None"
## [8] "1"
## [9] NA
```

Others wrote “1” as their educational level. The team decided to lump these cases into a common category called the uninformative responses.

Recode uninformative marital status

```r
MyFilteredData <- MyFilteredData %>%
  mutate(Dem_maritalstatus =
           case_when(Dem_maritalstatus %in% c("5", "Other or would rather not say")
                     ~ "Undisclosed/Uninformative response",
                     TRUE ~ Dem_maritalstatus))
```

Recode education level & mom's education level

The response options for the two questions on one's educational level and their mom's educational level included unnecessary hyphens. I removed those from the responses using the `str_remove()` function from the stringr package

```r
MyFilteredData$Dem_edu <- str_remove(MyFilteredData$Dem_edu, "- ")
MyFilteredData$Dem_edu_mom <- str_remove(MyFilteredData$Dem_edu_mom, "- ")
```

- The first argument is the column that I want to remove the string from.
- The second argument is the string I want to remove (in this case, the hyphen following by a whitespace **"-"**)

I also recoded the **None and "1"** answers to show that these responses are uninformative.

```r
MyFilteredData <- MyFilteredData %>%
  mutate_at(.vars = c("Dem_edu", "Dem_edu_mom"),
            .funs = ~ case_when(.x %in% c("None", "1") ~ "None/Uninformative response",
                                TRUE ~ .x))
```

I used a slightly different function called `mutate_at` from the dplyr package to overwrite multiple columns that share similar arguments.

- In the first argument, I specified the columns I wanted to overwrite - `Dem_edu` and `Dem_edu_mom`.
- In the second argument, I specified the function I wanted to use. In this case, I used the `case_when()` function.
  - Note that I use the tilde (~) in front of the function `case_when()`.
  - In the arguments of the `case_when()` function, I used **.x** in place of the variable name as I dealt with multiple variables here.
  - Essentially, the arguments stated that if the values in either `Dem_edu` or `Dem_edu_mom` match **None** or **"1"**, R would replace these values with **None/Uninformative response**. Otherwise, R would keep the original values of these variables.

Convert scale responses to numeric

Since I worked with the textual version of the data, I needed to convert the textual encoding of participants' responses to standard Likert-scale items to numerics. I did so using similar syntax I had when recoding demographic variables.

```r
# Recode multiple columns that share common text in their names
MyFilteredData <- MyFilteredData %>%
  mutate_at(
  .vars = vars(contains("PSS10")),
  .funs = ~ case_when(.x == "Never" ~ 1,
                      .x == "Almost never" ~ 2,
                      .x == "Sometimes" ~ 3,
                      .x == "Fairly often" ~ 4,
                      .x == "Very often" ~ 5)
  )
```

```r
# Recode multiple columns that do not share any common text in their names
MyFilteredData <- MyFilteredData %>%
  mutate_at(
  .vars = vars(matches("Corona_concerns|Compliance|BFF|Distress|SPS|Coping|Expl_media")),
  .funs = ~case_when(.x == "Strongly disagree" ~ 1,
                    .x == "Disagree" ~ 2,
                    .x == "Slightly disagree" ~ 3,
                    .x == "Slightly agree" ~ 4,
                    .x == "Agree" ~ 5,
                    .x == "Strongly agree" ~ 6)
  )
```

```r
# Recode the Trust_countryrmeasure
MyFilteredData <- MyFilteredData %>%
  mutate(Trust_countrymeasure = case_when(
    Trust_countrymeasure == "Too little" ~ 0,
    Trust_countrymeasure == "Appropriate" ~ 5,
    Trust_countrymeasure == "Too much" ~ 10,
    TRUE ~ as.numeric(Trust_countrymeasure)))
```

```
## Warning in eval_tidy(pair$rhs, env = default_env): NAs introduced by coercion
```

Export cleaned data to a separate file for analysis

```r
write.csv(MyFilteredData, "COVIDiSTRESS_cleaned.csv", row.names = FALSE)
```

I used the `write.csv()` function to export the cleaned dataset `MyFilteredData` into a cleaned csv file for further analysis.

- The first argument is the name of the data object in R.
- The second argument is the name of the new data file I wanted to save. Note that this will be automatically saved in the same folder that this Markdown file is at.
- The last argument `row.name = FALSE` tells R to not add in an additional column numbering each of the observations in our dataset.

Conclusion

This data cleaning file demonstrates how researchers can use a programming language like R to document every step of their data cleaning process. The contents included in this demonstration notebook is by no mean exhaustive, but researchers can use what I have here as a start when they begin their data cleaning in R. Alongside detailed reports of the data exploratory and data analysis stages, researchers are able to make their work completely transparent and reproducible. Generating a step-by-step document on how the research is done can be a time-consuming and daunting task, but it is highly rewarding. This contributes to the advance of science by motivating researchers to re-evaluate past work, potentially propose a better approach to look at a given problem, and easily building on previous studies.