Data Wrangling

0 - Info about the data and code

The following code will load in raw data from the Parenting Across Cultures data set and wrangle it. The final product will then be a wrangled rds file that can be used for subsequent analyses.

1 - Loading libraries

```r
# Loading haven to read the SPSS file with the raw data into R
library(haven)
# Loading tidyverse to get dplyr to handle data frames more easily
library(tidyverse)
```

```
Warning: package 'ggplot2' was built under R version 4.3.3

Warning: package 'tidyr' was built under R version 4.3.3

Warning: package 'readr' was built under R version 4.3.3

Warning: package 'purrr' was built under R version 4.3.3

Warning: package 'lubridate' was built under R version 4.3.3

── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
  dplyr     1.1.4     readr     2.1.5
  forcats   1.0.0     stringr   1.5.1
  ggplot2   3.5.1     tibble    3.2.1
  lubridate 1.9.4     tidyr     1.3.1
  purrr     1.0.4
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
  dplyr::filter() masks stats::filter()
  dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
# Loading labelled to unclass the special Haven objects (double with label) to factors without having to specify each column
library(labelled)
```

```
Warning: package 'labelled' was built under R version 4.3.3
```

2 - Loading data set

```r
# Loading data into R from the SPSS file
raw_df <- read_sav("Data_raw.sav")
```

3 - Getting to Know the Variables

There are 1454 observations in the data set and 2814 columns/variables in the data set.

When looking at the column names, there are several abbreviations that can be looked up in the code book. Here are the meaning of these abbreviations:

- DCBC - Child behavior checklist - Father
- MCBC - Child behavior checklist - Mother
- YSR - Child behavior checklist - Child self-report
- YASR - Young adult self report (similar questions as in the child behavior checklist)
- CvFROH - Parental acceptance-rejection/control questionnaire - Youth referencing father
- CvMROH - Parental acceptance-rejection/control questionnaire - youth referencing mother
- DROH - Parental acceptance-rejection/control questionnaire - father
- MROH - Parental acceptance-rejection/control questionnaire - mother
- fx/mx/cx + rohaff/rohhos/rohneg/rohrej/rohcon - Parental acceptance-rejection/control questionnaire subscales
- where x designates the wave
- fx/mx/cx + with/som/anx/del/agg/int/ext - Child behavior checklist subscales
- where x is designating the wave
- CPBS - Childhood prosocial behavior scale - Youth
- AAAM/F_Y or AAAxRM/F_Y - Adapted adult attachment scale - Youth
- where x is the item number
- AAAx_F - Adapted adult attachment scale - Father
- AAAx_M - Adapted adult attachment scale - Mother
- RBS - has no supplied full name in the code book, but it could be something like a risky behavior scale - Youth

Other than that, it seems like the sufix yx, where x is a number, designates the wave of the measurement. This is true for all waves except the first wave, which does not have any sufix. Another observation, is that there are various names for the variable giving the age of the child at different ages.

To get to know the data better, I used the wave sufix (yx) to find the measurements at each wave. I used the following regex, where the number can be switched out with the wave of interest to explore the measures collected at that age:

```r
# Looking for variable names that start with a letter (the variables will start with y, m, f, or c), then a number (in this case 10), then at least one more character
grep("^([[:alpha:]]10[[:alpha:]]+)", colnames(raw_df), value = TRUE)
```

```
 [1] "y10AAA1_M"    "y10AAA2_M"    "y10AAA3_M"    "y10AAA4_M"    "y10AAA5_M"   
 [6] "y10AAA6_M"    "y10AAA7_M"    "y10AAA8_M"    "y10AAA9_M"    "y10AAA10_M"  
[11] "y10AAAB_M"    "y10AAA11_M"   "y10AAA12_M"   "y10AAA13_M"   "y10AAA14_M"  
[16] "y10AAA15_M"   "y10AAA16_M"   "y10AAA17_M"   "y10AAA18_M"   "y10AAA19_M"  
[21] "y10AAA20_M"   "y10AAA1RM_Y"  "y10AAA2RM_Y"  "y10AAA3RM_Y"  "y10AAA4RM_Y" 
[26] "y10AAA5RM_Y"  "y10AAA6RM_Y"  "y10AAA7RM_Y"  "y10AAA8RM_Y"  "y10AAA9RM_Y" 
[31] "y10AAA10RM_Y" "y10AAAF_Y"    "y10AAA1RF_Y"  "y10AAA2RF_Y"  "y10AAA3RF_Y" 
[36] "y10AAA4RF_Y"  "y10AAA5RF_Y"  "y10AAA6RF_Y"  "y10AAA7RF_Y"  "y10AAA8RF_Y" 
[41] "y10AAA9RF_Y"  "y10AAA10RF_Y" "y10YASR1"     "y10YASR2"     "y10YASR3"    
[46] "y10YASR4"     "y10YASR5"     "y10YASR6"     "y10YASR7"     "y10YASR8"    
[51] "y10YASR9"     "y10YASR10"    "y10YASR11"    "y10YASR12"    "y10YASR13"   
[56] "y10YASR14"    "y10YASR15"    "y10YASR16"    "y10YASR17"    "y10YASR18"   
[61] "y10YASR19"    "y10YASR20"    "y10YASR21"    "y10YASR22"    "y10YASR23"   
[66] "y10YASR24"    "y10YASR25"    "y10YASR26"    "y10YASR27"    "y10YASR28"   
[71] "y10YASR29"    "y10YASR30"    "y10YASR31"    "y10YASR32"    "y10YASR33"   
[76] "y10YASR34"    "y10YASR35"    "y10YASR36"    "y10YASR37"    "y10YASR38"   
```

```
 [81] "y10YASR39"   "y10YASR40"   "y10YASR41"   "y10YASR42"   "y10YASR43"
 [86] "y10YASR44"   "y10YASR45"   "y10YASR46"   "y10YASR47"   "y10YASR48"
 [91] "y10YASR49"   "y10YASR50"   "y10YASR51"   "y10YASR52"   "y10AAA1_F"
 [96] "y10AAA2_F"   "y10AAA3_F"   "y10AAA4_F"   "y10AAA5_F"   "y10AAA6_F"
[101] "y10AAA7_F"   "y10AAA8_F"   "y10AAA9_F"   "y10AAA10_F"  "y10AAAB_F"
[106] "y10AAA11_F"  "y10AAA12_F"  "y10AAA13_F"  "y10AAA14_F"  "y10AAA15_F"
[111] "y10AAA16_F"  "y10AAA17_F"  "y10AAA18_F"  "y10AAA19_F"  "y10AAA20_F"
[116] "c10with"     "c10anx"      "c10int"      "c10tru"      "c10del"
[121] "c10agg"      "c10ext"
```

After doing across all the waves, I found that the following measures were collected at each wave:

- y1 - DCBC, MCBC, YSR, CvFROH, CvMROH, DROH, MROH, fx/mx/cxrohaff/rohhos/rohneg/rohrej/rohcon, fx/mx/cxwith/som/anx/del/agg/int/ext
- y2 - DCBC, MCBC, YSR, CvFROH, CvMROH, DROH, MROH, fx/mx/cxrohaff/rohhos/rohneg/rohrej/rohcon, fx/mx/cxwith/som/anx/del/agg/int/ext
- y3 - DCBC, MCBC, YSR, CvFROH, CvMROH, DROH, MROH, fx/mx/cxrohaff/rohhos/rohneg/rohrej/rohcon, fx/mx/cxwith/som/anx/del/agg/int/ext, CPBS
- y4 - none
- y5 - DCBC, MCBC, YSR, CvFROH, CvMROH, DROH, MROH, fx/mx/cxrohaff/rohhos/rohneg/rohrej/rohcon, fx/mx/cxwith/som/anx/del/agg/int/ext
- y6 - DCBC, MCBC, DROH, MROH, fx/mxrohaff/rohhos/rohneg/rohrej/rohcon, fx/mxwith/som/anx/del/agg/int/ext
- y7 - DCBC, MCBC, YSR, CvFROH, CvMROH, fx/mx/cxrohaff/rohhos/rohneg/rohrej/rohcon, fx/mx/cxwith/som/anx/del/agg/int/ext
- y8 - DCBC, MCBC, YSR, CvFROH, CvMROH, fx/mx/cxwith/som/anx/del/agg/int/ext
- y9 - DCBC, MCBC, fx/mxwith/som/anx/del/agg/int/ext
- y10 - AAAM/F_Y or AAAxRM/F_Y, AAAx_F, AAAx_M, YASR, cxwith/som/anx/del/agg/int/ext
- y11 - DCBC, MCBC, YASR, fx/mx/cxwith/som/anx/del/agg/int/ext
- y12 - none
- y13 - DCBC, MCBC, YASR, fx/mx/cxwith/som/anx/del/agg/int/ext, RBS

It looks like we can look at rater disagreement at waves 1, 2, 3, 5, 7, 8, 11, and 13, as they all have recordings of DCBC, MCBC, and YSR/YASR.

Now that we know more about the variables in the data frame, we are reay to start data wrangling. However, before doing any actual change to the data frame, I will create a copy of it, so that I can freely edit it and always have a back copy of the data.

```
# Copying the raw data frame to a data frame we can change, called data_df
data_df <- raw_df
```

4 - Looking for out-of-bounds values

The CBLC, YSR, and YASR are all scored on a scale between 0 and 2. However, in data analyses, there were a few values that came up that were out of this range. I will therefore look for values more than 2 in these scales and convert them to missing values:

```
data_df <- data_df %>% mutate(
  across(matches("DCBC|YSR|YASR"), ~ if_else(.x > 2, NA, .x))
)
```

4 - Cleaning up variable names for age

As noted earlier, it looks like the age variables are named differently across the years. However, most seem to have the word "age" in it, so we can use a regex to find all the age variables:

```
# Looking for any variable name containing "Age" or "age"
grep("[Aa]ge", colnames(data_df), value = TRUE)
```

```
[1] "y1AgeInt"          "y2AgeIntChild"    "y3AgeIntChild"    "y4AgeIntChild"
[5] "y5AgeIntChild"     "y6AgePACIntChild" "y7AgeIntChild"    "y9ChildAge"
[9] "C11IntAge"
```

It looks like we are missing the age variable for waves 8, 10, and 13. I went over the variables for these waves manually to see whether they were named something else not captured by the regex, but it doesn't seem like our data set contains the age of the children at these waves. We will deal with this later.

The missing age variables are in a separate data file which I will read in and add into this data set.

```
# Loading the age data into R from the SPSS file
age_df <- read_sav("Years8_10_13_ChAge.sav")
```

Adding the variables from the age data frame to the data_df data frame:

```
data_df <- merge(data_df, age_df, by = "ID")
```

Checking whether the columns got added:

```
# Looking for any variable name containing "Age" or "age"
grep("[Aa]ge", colnames(data_df), value = TRUE)
```

```
 [1] "y1AgeInt"          "y2AgeIntChild"    "y3AgeIntChild"    "y4AgeIntChild"
 [5] "y5AgeIntChild"     "y6AgePACIntChild" "y7AgeIntChild"    "y9ChildAge"
 [9] "C11IntAge"         "y8ChildAge"       "y10ChildAge"      "y13ChildAge"
```

For consistency (and ease when converting the data from wide to long later), I will rename the age variables to have the same name, with only the wave number being different:

```
# Looking for variable names that start with a character, then at least one digit (in a capture
group), some optional characters, the word "Age" or "age", followed by some optional characters.
Then, the names are changed to y, the number taken from the capture group (to preserve the wave
number for each variable), and the text "Age"
colnames(data_df) <- gsub("^\\w(\\d+)(?:.*)?[Aa]ge(?:.*)?", "y\\1Age", colnames(data_df))

# Printing the new age variable names by looking for variables with the words "Age" or "age" in
them
grep("[Aa]ge", colnames(data_df), value = TRUE)
```

```
 [1] "y1Age"  "y2Age"  "y3Age"  "y4Age"  "y5Age"  "y6Age"  "y7Age"  "y9Age" 
 [9] "y11Age" "y8Age"  "y10Age" "y13Age"
```

5 - Cleaning up variable names for waves

Using regex to find the variables that don’t have a number indicating the wave, which means they belong to wave 1:

```
# Looking for variable names that start with a at least three letters, then at least one number
grep("^([[:alpha:]]{3,}\\d+)", colnames(data_df), value = TRUE)
```

```
  [1] "DCBC1"      "DCBC2"      "DCBC3"      "DCBC4"      "DCBC5"     
  [6] "DCBC6"      "DCBC7"      "DCBC8"      "DCBC9"      "DCBC10"    
 [11] "DCBC11"     "DCBC12"     "DCBC13"     "DCBC14"     "DCBC15"    
 [16] "DCBC16"     "DCBC17"     "DCBC18"     "DCBC19"     "DCBC20"    
 [21] "DCBC21"     "DCBC22"     "DCBC23"     "DCBC24"     "DCBC25"    
 [26] "DCBC26"     "DCBC27A"    "DCBC27B"    "DCBC27C"    "DCBC27D"   
 [31] "DCBC27DEXP" "DCBC27E"    "DCBC27F"    "DCBC27G"    "DCBC28"    
 [36] "DCBC29"     "DCBC30"     "DCBC31"     "DCBC32"     "DCBC33"    
 [41] "DCBC34"     "DCBC35"     "DCBC36"     "DCBC37"     "DCBC38"    
 [46] "DCBC39"     "DCBC40"     "DCBC41"     "DCBC42"     "DCBC43"    
 [51] "DCBC44"     "DCBC45"     "DCBC46"     "DCBC47"     "DCBC48"    
 [56] "DCBC49"     "DCBC50"     "DCBC51"     "DCBC52"     "DCBC53"    
 [61] "DCBC54"     "DCBC55"     "DCBC55EXP"  "DCBC56"     "DCBC57"    
 [66] "DCBC58"     "MCBC1"      "MCBC2"      "MCBC3"      "MCBC4"     
 [71] "MCBC5"      "MCBC6"      "MCBC7"      "MCBC8"      "MCBC9"     
 [76] "MCBC10"     "MCBC11"     "MCBC12"     "MCBC13"     "MCBC14"    
 [81] "MCBC15"     "MCBC16"     "MCBC17"     "MCBC18"     "MCBC19"    
 [86] "MCBC20"     "MCBC21"     "MCBC22"     "MCBC23"     "MCBC24"    
 [91] "MCBC25"     "MCBC26"     "MCBC27A"    "MCBC27B"    "MCBC27C"   
 [96] "MCBC27D"    "MCBC27DEXP" "MCBC27E"    "MCBC27F"    "MCBC27G"   
[101] "MCBC28"     "MCBC29"     "MCBC30"     "MCBC31"     "MCBC32"    
[106] "MCBC33"     "MCBC34"     "MCBC35"     "MCBC36"     "MCBC37"    
[111] "MCBC38"     "MCBC39"     "MCBC40"     "MCBC41"     "MCBC42"    
[116] "MCBC43"     "MCBC44"     "MCBC45"     "MCBC46"     "MCBC47"    
[121] "MCBC48"     "MCBC49"     "MCBC50"     "MCBC51"     "MCBC52"    
[126] "MCBC53"     "MCBC54"     "MCBC55"     "MCBC55EXP"  "MCBC56"    
[131] "MCBC57"     "MCBC58"     "YSR1"       "YSR2"       "YSR3"      
[136] "YSR4"       "YSR5"       "YSR6"       "YSR7"       "YSR8"      
[141] "YSR9"       "YSR10"      "YSR11"      "YSR12"      "YSR13"     
[146] "YSR14"      "YSR15"      "YSR16"      "YSR17"      "YSR18"     
[151] "YSR19"      "YSR20"      "YSR21"      "YSR22"      "YSR23"     
[156] "YSR24"      "YSR25"      "YSR26A"     "YSR26B"     "YSR26C"    
[161] "YSR26D"     "YSR26DEXP"  "YSR26E"     "YSR26F"     "YSR26G"    
[166] "YSR27"      "YSR28"      "YSR29"      "YSR30"      "YSR31"     
```

```
[171] "YSR32"      "YSR33"      "YSR34"      "YSR35"      "YSR36"
[176] "YSR37"      "YSR38"      "YSR39"      "YSR40"      "YSR41"
[181] "YSR42"      "YSR43"      "YSR44"      "YSR45"      "YSR46"
[186] "YSR47"      "YSR48"      "YSR49"      "YSR50"      "YSR51"
[191] "YSR51A"     "YSR52"      "YSR53"      "CVFROH1"    "CVFROH2"
[196] "CVFROH3"    "CVFROH4"    "CVFROH5"    "CVFROH6"    "CVFROH7"
[201] "CVFROH8"    "CVFROH9"    "CVFROH10"   "CVFROH11"   "CVFROH12"
[206] "CVFROH13"   "CVFROH14"   "CVFROH15"   "CVFROH16"   "CVFROH17"
[211] "CVFROH18"   "CVFROH19"   "CVFROH20"   "CVFROH21"   "CVFROH22"
[216] "CVFROH23"   "CVFROH24"   "CVFROH25"   "CVFROH26"   "CVFROH27"
[221] "CVFROH28"   "CVFROH29"   "CVMROH1"    "CVMROH2"    "CVMROH3"
[226] "CVMROH4"    "CVMROH5"    "CVMROH6"    "CVMROH7"    "CVMROH8"
[231] "CVMROH9"    "CVMROH10"   "CVMROH11"   "CVMROH12"   "CVMROH13"
[236] "CVMROH14"   "CVMROH15"   "CVMROH16"   "CVMROH17"   "CVMROH18"
[241] "CVMROH19"   "CVMROH20"   "CVMROH21"   "CVMROH22"   "CVMROH23"
[246] "CVMROH24"   "CVMROH25"   "CVMROH26"   "CVMROH27"   "CVMROH28"
[251] "CVMROH29"   "DROH1"      "DROH2"      "DROH3"      "DROH4"
[256] "DROH5"      "DROH6"      "DROH7"      "DROH8"      "DROH9"
[261] "DROH10"     "DROH11"     "DROH12"     "DROH13"     "DROH14"
[266] "DROH15"     "DROH16"     "DROH17"     "DROH18"     "DROH19"
[271] "DROH20"     "DROH21"     "DROH22"     "DROH23"     "DROH24"
[276] "DROH25"     "DROH26"     "DROH27"     "DROH28"     "DROH29"
[281] "MROH1"      "MROH2"      "MROH3"      "MROH4"      "MROH5"
[286] "MROH6"      "MROH7"      "MROH8"      "MROH9"      "MROH10"
[291] "MROH11"     "MROH12"     "MROH13"     "MROH14"     "MROH15"
[296] "MROH16"     "MROH17"     "MROH18"     "MROH19"     "MROH20"
[301] "MROH21"     "MROH22"     "MROH23"     "MROH24"     "MROH25"
[306] "MROH26"     "MROH27"     "MROH28"     "MROH29"
```

The following code, by requiring 3 or more letters, then a number, only includes the variables that are items or scales. It excludes columns with names that are not general for all waves, such as “ID”, “ChGender”, “IDN”, and so on. Consequently, all the column names from the above regex belongs to wave 1 and should therefore get the prefix of “y1”. The consistency in the wave numbering in the variable will make the wide to long conversion easier later.

Adding the prefix “y1” to all the variables indicated by the above code:

```
# Looking for variable names that start with a at least three letters, then at least one number.
That entire variable name is captured in a capture group. Then we substitute the variable names
with the prefix "y1" followed by the old variable name that was capture in the capture group
colnames(data_df) <- gsub("^([[:alpha:]]{3,}\\d+)", "y1\\1", colnames(data_df))
```

Checking that the column name change was successful for wave 1:

```
# Looking for variable names that start with a letter, then the number 1, then another letter
grep("^([[:alpha:]]1[[:alpha:]])", colnames(data_df), value = TRUE)
```

```
 [1] "m1educ"      "f1educ"      "y1DCBC1"     "y1DCBC2"     "y1DCBC3"
 [6] "y1DCBC4"     "y1DCBC5"     "y1DCBC6"     "y1DCBC7"     "y1DCBC8"
[11] "y1DCBC9"     "y1DCBC10"    "y1DCBC11"    "y1DCBC12"    "y1DCBC13"
[16] "y1DCBC14"    "y1DCBC15"    "y1DCBC16"    "y1DCBC17"    "y1DCBC18"
[21] "y1DCBC19"    "y1DCBC20"    "y1DCBC21"    "y1DCBC22"    "y1DCBC23"
[26] "y1DCBC24"    "y1DCBC25"    "y1DCBC26"    "y1DCBC27A"   "y1DCBC27B"
[31] "y1DCBC27C"   "y1DCBC27D"   "y1DCBC27DEXP" "y1DCBC27E"  "y1DCBC27F"
```

```
 [36] "y1DCBC27G"    "y1DCBC28"     "y1DCBC29"     "y1DCBC30"     "y1DCBC31"    
 [41] "y1DCBC32"     "y1DCBC33"     "y1DCBC34"     "y1DCBC35"     "y1DCBC36"    
 [46] "y1DCBC37"     "y1DCBC38"     "y1DCBC39"     "y1DCBC40"     "y1DCBC41"    
 [51] "y1DCBC42"     "y1DCBC43"     "y1DCBC44"     "y1DCBC45"     "y1DCBC46"    
 [56] "y1DCBC47"     "y1DCBC48"     "y1DCBC49"     "y1DCBC50"     "y1DCBC51"    
 [61] "y1DCBC52"     "y1DCBC53"     "y1DCBC54"     "y1DCBC55"     "y1DCBC55EXP" 
 [66] "y1DCBC56"     "y1DCBC57"     "y1DCBC58"     "y1MCBC1"      "y1MCBC2"     
 [71] "y1MCBC3"      "y1MCBC4"      "y1MCBC5"      "y1MCBC6"      "y1MCBC7"     
 [76] "y1MCBC8"      "y1MCBC9"      "y1MCBC10"     "y1MCBC11"     "y1MCBC12"    
 [81] "y1MCBC13"     "y1MCBC14"     "y1MCBC15"     "y1MCBC16"     "y1MCBC17"    
 [86] "y1MCBC18"     "y1MCBC19"     "y1MCBC20"     "y1MCBC21"     "y1MCBC22"    
 [91] "y1MCBC23"     "y1MCBC24"     "y1MCBC25"     "y1MCBC26"     "y1MCBC27A"   
 [96] "y1MCBC27B"    "y1MCBC27C"    "y1MCBC27D"    "y1MCBC27DEXP" "y1MCBC27E"   
[101] "y1MCBC27F"    "y1MCBC27G"    "y1MCBC28"     "y1MCBC29"     "y1MCBC30"    
[106] "y1MCBC31"     "y1MCBC32"     "y1MCBC33"     "y1MCBC34"     "y1MCBC35"    
[111] "y1MCBC36"     "y1MCBC37"     "y1MCBC38"     "y1MCBC39"     "y1MCBC40"    
[116] "y1MCBC41"     "y1MCBC42"     "y1MCBC43"     "y1MCBC44"     "y1MCBC45"    
[121] "y1MCBC46"     "y1MCBC47"     "y1MCBC48"     "y1MCBC49"     "y1MCBC50"    
[126] "y1MCBC51"     "y1MCBC52"     "y1MCBC53"     "y1MCBC54"     "y1MCBC55"    
[131] "y1MCBC55EXP"  "y1MCBC56"     "y1MCBC57"     "y1MCBC58"     "y1YSR1"      
[136] "y1YSR2"       "y1YSR3"       "y1YSR4"       "y1YSR5"       "y1YSR6"      
[141] "y1YSR7"       "y1YSR8"       "y1YSR9"       "y1YSR10"      "y1YSR11"     
[146] "y1YSR12"      "y1YSR13"      "y1YSR14"      "y1YSR15"      "y1YSR16"     
[151] "y1YSR17"      "y1YSR18"      "y1YSR19"      "y1YSR20"      "y1YSR21"     
[156] "y1YSR22"      "y1YSR23"      "y1YSR24"      "y1YSR25"      "y1YSR26A"    
[161] "y1YSR26B"     "y1YSR26C"     "y1YSR26D"     "y1YSR26DEXP"  "y1YSR26E"    
[166] "y1YSR26F"     "y1YSR26G"     "y1YSR27"      "y1YSR28"      "y1YSR29"     
[171] "y1YSR30"      "y1YSR31"      "y1YSR32"      "y1YSR33"      "y1YSR34"     
[176] "y1YSR35"      "y1YSR36"      "y1YSR37"      "y1YSR38"      "y1YSR39"     
[181] "y1YSR40"      "y1YSR41"      "y1YSR42"      "y1YSR43"      "y1YSR44"     
[186] "y1YSR45"      "y1YSR46"      "y1YSR47"      "y1YSR48"      "y1YSR49"     
[191] "y1YSR50"      "y1YSR51"      "y1YSR51A"     "y1YSR52"      "y1YSR53"     
[196] "y1CVFROH1"    "y1CVFROH2"    "y1CVFROH3"    "y1CVFROH4"    "y1CVFROH5"   
[201] "y1CVFROH6"    "y1CVFROH7"    "y1CVFROH8"    "y1CVFROH9"    "y1CVFROH10"  
[206] "y1CVFROH11"   "y1CVFROH12"   "y1CVFROH13"   "y1CVFROH14"   "y1CVFROH15"  
[211] "y1CVFROH16"   "y1CVFROH17"   "y1CVFROH18"   "y1CVFROH19"   "y1CVFROH20"  
[216] "y1CVFROH21"   "y1CVFROH22"   "y1CVFROH23"   "y1CVFROH24"   "y1CVFROH25"  
[221] "y1CVFROH26"   "y1CVFROH27"   "y1CVFROH28"   "y1CVFROH29"   "y1CVMROH1"   
[226] "y1CVMROH2"    "y1CVMROH3"    "y1CVMROH4"    "y1CVMROH5"    "y1CVMROH6"   
[231] "y1CVMROH7"    "y1CVMROH8"    "y1CVMROH9"    "y1CVMROH10"   "y1CVMROH11"  
[236] "y1CVMROH12"   "y1CVMROH13"   "y1CVMROH14"   "y1CVMROH15"   "y1CVMROH16"  
[241] "y1CVMROH17"   "y1CVMROH18"   "y1CVMROH19"   "y1CVMROH20"   "y1CVMROH21"  
[246] "y1CVMROH22"   "y1CVMROH23"   "y1CVMROH24"   "y1CVMROH25"   "y1CVMROH26"  
[251] "y1CVMROH27"   "y1CVMROH28"   "y1CVMROH29"   "y1DROH1"      "y1DROH2"     
[256] "y1DROH3"      "y1DROH4"      "y1DROH5"      "y1DROH6"      "y1DROH7"     
[261] "y1DROH8"      "y1DROH9"      "y1DROH10"     "y1DROH11"     "y1DROH12"    
[266] "y1DROH13"     "y1DROH14"     "y1DROH15"     "y1DROH16"     "y1DROH17"    
[271] "y1DROH18"     "y1DROH19"     "y1DROH20"     "y1DROH21"     "y1DROH22"    
[276] "y1DROH23"     "y1DROH24"     "y1DROH25"     "y1DROH26"     "y1DROH27"    
[281] "y1DROH28"     "y1DROH29"     "y1MROH1"      "y1MROH2"      "y1MROH3"     
[286] "y1MROH4"      "y1MROH5"      "y1MROH6"      "y1MROH7"      "y1MROH8"     
[291] "y1MROH9"      "y1MROH10"     "y1MROH11"     "y1MROH12"     "y1MROH13"    
[296] "y1MROH14"     "y1MROH15"     "y1MROH16"     "y1MROH17"     "y1MROH18"    
[301] "y1MROH19"     "y1MROH20"     "y1MROH21"     "y1MROH22"     "y1MROH23"    
```

```
[306] "y1MROH24"     "y1MROH25"     "y1MROH26"     "y1MROH27"     "y1MROH28"
[311] "y1MROH29"     "m1rohaff"     "m1rohhos"     "m1rohneg"     "m1rohrej"
[316] "m1rohcon"     "f1rohaff"     "f1rohhos"     "f1rohneg"     "f1rohrej"
[321] "f1rohcon"     "c1mroaff"     "c1mrohos"     "c1mroneg"     "c1mrorej"
[326] "c1mrocon"     "c1froaff"     "c1frohos"     "c1froneg"     "c1frorej"
[331] "c1frocon"     "m1with"       "m1som"        "m1anx"        "m1del"
[336] "m1agg"        "m1int"        "m1ext"        "f1with"       "f1som"
[341] "f1anx"        "f1del"        "f1agg"        "f1int"        "f1ext"
[346] "c1with"       "c1som"        "c1anx"        "c1del"        "c1agg"
[351] "c1int"        "c1ext"        "y1Age"
```

It looks like all of the variables for wave 1 now has the y1 prefix. As a last double-check, I will look up all the variable names that do not have a wave, to make sure we didn't miss a variable that should have a wave 1 prefix. In the following regex, we are taking advantage of the fact that all the wave variables have the prefix yx, mx, fx, cx, where x is a number, by saying that the variable name needs to start with at least two letters (i.e. no numbers).

```r
# Looking for variable names that start with at least two letters
grep("^([[:alpha:]]{2,})", colnames(data_df), value = TRUE)
```

```
[1] "ID"       "IDN"      "ChGender"
```

As can be seen with these variables, the addition of the y1 prefix for all the variables for wave 1 seems to have been successful.

6 - Wide to long conversion

Adding the rater prefix as a suffix to the variable names in addition to a prefix. This will preserve the rater information in the variable name when the first two characters of the variable names are removed and the wave number converted to rows in the wide to long conversion.

```r
# Looking for variable names starting with m, f, or c (in a capture group), then at least one number, then at least one character. All of this is captured in another capture group. Then the variable names are replaced by the whole old variable name and the new sufix of the rater
colnames(data_df) <- gsub("^(([mfc])\\d+[[:alpha:]]+)", "\\1_\\2", colnames(data_df))
```

The wide to long conversion is taking the variables for individual waves and combining them so that each row is a wave instead. There will still be different variables for the various raters.

```r
# Takes 3 minutes to run through
# Taking all the variables except those that are general to all of the waves. In names_to, we are saving that we want one new variable called wave and the rest of the new variables are the old variables without the wave information. This information is taken from the capture groups in the regex in names_pattern, which looks for variable names starting with a character, then at least one digit (to allow for both one-digit and two-digit numbers, e.g. 1 and 10), which is in the first capture group, then more characters that are in a second capture group. The wave number from the first capture group becomes the value for the wave variable, while the second capture group (variable name without wave number) becomes the new variable name
data_df_long <- data_df %>%
  pivot_longer(-c("ID",  "IDN", "m1educ_m", "f1educ_f", "ChGender"), names_to = c("Wave", ".value"), names_pattern = "^[[:alpha:]](\\d+)(.*)")
```

This wide to long conversion produces a lot of warnings as the label of the item (the wording of the item) differ across several of the waves. The warnings are currently suppressed, since they took up a lot of space. However, be aware that the item label will reflect the wording at the earliest wave, and that one should look to the code book for the specific wording at each wave.

Otherwise, it looks like the wide to long conversion was successful. For example, as a check, there are 12 waves in total, 1-11 and 13. We had 1454 observations in the data_df and we now have 17 448 observations in the long data set. This should be equal to the original number of observations times the number of waves. We can check this with 1 454 * 12 = 17 448, so it seems like we have the correct number of observations in the long data set. Manually looking through the new data frames also makes it seem like the conversion was successful.

7 - Calculating scales from items

These items are tentative, as they might change after a discussion of how much difference to allow between the versions.

We will only use the items that have similar wordings across the various raters. Consequently, we cannot use the original internalizing and externalizing scales, but will instead calculate our own with the items that are similar across raters only. Otherwise, the items that are included in the internalizing or externalizing scales, are the same as designated in the code book for the child behavior checklist.

One of the items, the one about stealing, is one item in the young adult version, but two items in the child and parent version. I will take the highest rating of the two items for the child and parent version and use that to compare to the one item in the young adult version. This requires calculating a new variable for the max value from the two items:

```
# Creating a new variable that is the max value from either item 39 or 40 for the parent and child version of CBCL
data_df_long <- data_df_long %>%
  mutate(
  DCBC_max_39_40 = case_when(
    DCBC39 == pmax(DCBC39, DCBC40) ~ DCBC39,
    DCBC40 == pmax(DCBC39, DCBC40) ~ DCBC40
  ), MCBC_max_39_40 = case_when(
    MCBC39 == pmax(MCBC39, MCBC40) ~ MCBC39,
    MCBC40 == pmax(MCBC39, MCBC40) ~ MCBC40
  ), YSR_max_37_38 = case_when(
    YSR37 == pmax(YSR37, YSR38) ~ YSR37,
    YSR38 == pmax(YSR37, YSR38) ~ YSR38
  )
)
```

The item numbers for each item depend slightly based on the rater version. Therefore, we need different item vectors for internalizing and externalizing symptoms for each rater. These will be used later when actually combining the items into scales.

```
# Defining vectors with the items for each rater for the externalizing scales
ext_scale_item_f_vect <- c("DCBC19", "DCBC21", "DCBC1", "DCBC2", "DCBC5", "DCBC6", "DCBC8", "DCBC18", "DCBC28", "DCBC32", "DCBC36", "DCBC42", "DCBC46", "DCBC47", "DCBC48", "DCBC50", "DCBC54", "DCBC_max_39_40")
ext_scale_item_m_vect <- c("MCBC19", "MCBC21", "MCBC1", "MCBC2", "MCBC5", "MCBC6", "MCBC8", "MCBC18", "MCBC28", "MCBC32", "MCBC36", "MCBC42", "MCBC46", "MCBC47", "MCBC48", "MCBC50", "MCBC54", "MCBC_max_39_40")
ext_scale_item_c_vect <- c("YSR18", "YSR20", "YSR1", "YSR2", "YSR5", "YSR6", "YSR8", "YSR17",
"YSR27", "YSR31", "YSR35", "YSR40", "YSR43", "YSR44", "YSR45", "YSR46", "YSR50", "YSR_max_37_38")
ext_scale_item_ya_vect <- c("YASR1", "YASR3", "YASR7", "YASR8", "YASR9", "YASR19", "YASR21",
"YASR23", "YASR29", "YASR32", "YASR35", "YASR39", "YASR42", "YASR43", "YASR44", "YASR45", "YASR47",
"YASR37")

# Defining vectors with the items for each rater for the internalizing scales
int_scale_item_f_vect <- c("DCBC30", "DCBC33", "DCBC37", "DCBC53", "DCBC57", "DCBC3", "DCBC4",
"DCBC13", "DCBC14", "DCBC15", "DCBC16", "DCBC17", "DCBC22", "DCBC23", "DCBC25", "DCBC34", "DCBC58")
  # DCBD53 is only mentioned once, even though it is in both the withdrawn and anxious/depressed
subscale
int_scale_item_m_vect <- c("MCBC30", "MCBC33", "MCBC37", "MCBC53", "MCBC57", "MCBC3", "MCBC4",
"MCBC13", "MCBC14", "MCBC15", "MCBC16", "MCBC17", "MCBC22", "MCBC23", "MCBC25", "MCBC34", "MCBC58")
  # MCBD53 is only mentioned once, even though it is in both the withdrawn and anxious/depressed
subscale
int_scale_item_c_vect <- c("YSR29", "YSR32", "YSR36", "YSR49", "YSR52", "YSR3", "YSR4", "YSR12",
"YSR13", "YSR14", "YSR15", "YSR16", "YSR21", "YSR22", "YSR24", "YSR33", "YSR53")
  # YSR49 is only mentioned once, even though it is in both the withdrawn and anxious/depressed
subscale
int_scale_item_ya_vect <- c("YASR4", "YASR6", "YASR14", "YASR15", "YASR16", "YASR17", "YASR18",
"YASR24", "YASR27", "YASR28", "YASR30", "YASR33", "YASR34", "YASR36", "YASR46", "YASR48", "YASR49")
```

```
# Creating new variables for the internalizing and externalizing scales for each rater, based on
the items in the vectors defined in the code above
data_df_long <- data_df_long %>%
  mutate(
    ext_scale_f = rowSums(data_df_long[, ext_scale_item_f_vect]),
    ext_scale_m = rowSums(data_df_long[, ext_scale_item_m_vect]),
    int_scale_f = rowSums(data_df_long[, int_scale_item_f_vect]),
    int_scale_m = rowSums(data_df_long[, int_scale_item_m_vect]),
    ext_scale_c  = case_when(
    # There was no child behavior checklist in wave 9, so it doesn't matter that there is no
condition where Wave == 9, it would get a NA value anyway
    Wave < 9 ~ rowSums(data_df_long[, ext_scale_item_c_vect]),
    Wave > 9 ~ rowSums(data_df_long[, ext_scale_item_ya_vect])
  ),
  int_scale_c  = case_when(
    Wave < 9 ~ rowSums(data_df_long[, int_scale_item_c_vect]),
    Wave > 9 ~ rowSums(data_df_long[, int_scale_item_ya_vect])
  ))
```

7.1 - Calculating Adapted Adult Attachment Scale at Wave 10

```
# One of the items, item 4 needs to be reverse coded, so I will make a new variable for AAA4FR_Y
and AAA4RM_Y
data_df_long <- data_df_long %>% mutate(AAA4RF_Y_R = case_when(
  AAA4RF_Y == 5 ~ 1,
  AAA4RF_Y == 4 ~ 2,
  AAA4RF_Y == 2 ~ 3,
  AAA4RF_Y == 1 ~ 5,
  .default = AAA4RF_Y
)) %>%
  mutate(AAA4RM_Y_R = case_when(
  AAA4RM_Y == 5 ~ 1,
  AAA4RM_Y == 4 ~ 2,
  AAA4RM_Y == 2 ~ 3,
  AAA4RM_Y == 1 ~ 5,
  .default = AAA4RM_Y
))

# Creating a vector with the items in the AAA questionnaire regarding father
aaa_scale_item_ya_father_vect <- c("AAA1RF_Y", "AAA2RF_Y", "AAA3RF_Y", "AAA4RF_Y_R", "AAA5RF_Y",
"AAA6RF_Y", "AAA7RF_Y", "AAA8RF_Y", "AAA9RF_Y", "AAA10RF_Y")

# Creating a vector with the items in the AAA questionnaire regarding mother
aaa_scale_item_ya_mother_vect <- c("AAA1RM_Y", "AAA2RM_Y", "AAA3RM_Y", "AAA4RM_Y_R", "AAA5RM_Y",
"AAA6RM_Y", "AAA7RM_Y", "AAA8RM_Y", "AAA9RM_Y", "AAA10RM_Y")

# Creating new variables for the AAA scale rated by the young adult in reference to mother and
father based on the items in the vectors defined above
data_df_long <- data_df_long %>%
  mutate(
    aaa_scale_ya_f = rowSums(data_df_long[, aaa_scale_item_ya_father_vect]),
    aaa_scale_ya_m = rowSums(data_df_long[, aaa_scale_item_ya_mother_vect]),
  )
```

Creating a variable that has children categorically divided into a low and high attachment group, based on a mean split:

```
data_df_long <- data_df_long %>%
  mutate(
    aaa_scale_ya_f_dich = case_when(
      aaa_scale_ya_f < median(data_df_long$aaa_scale_ya_f, na.rm = TRUE) ~ "Low",
      aaa_scale_ya_f > median(data_df_long$aaa_scale_ya_f, na.rm = TRUE) ~ "High",
    .default = NA),
    aaa_scale_ya_m_dich = case_when(
      aaa_scale_ya_m < median(data_df_long$aaa_scale_ya_m, na.rm = TRUE) ~ "Low",
      aaa_scale_ya_m > median(data_df_long$aaa_scale_ya_m, na.rm = TRUE) ~ "High",
      .default = NA),
    )
```

7.1 - Calculating RBS Scale at Wave 13

For some reason, there are only 4 out of the 9 items in the current data set, so we will use those to calculate the scale

```
# Creating a vector with the items for the RBS scale
rbs_scale_item_vect <- c("RBS1_Y", "RBS2_Y", "RBS5_Y", "RBS9_Y")

# Creating a new variables for the RBS scale based on the items in the vector defined above
data_df_long <- data_df_long %>%
  mutate(rbs_scale_ya = rowSums(data_df_long[, rbs_scale_item_vect]))
```

8 - More data cleaning

8.1 - Data types

First, there are several columns that have data that is of the data type double with a label. This data type works fairly well for now, but there might be other functions down the road that handles factors (which are more common) better than the labelled double. I will therefore use unlabelled() to convert those doubles with labels to factors. labelled() is easier to use than as.factor() as it automatically detects what variables to be converted to factors, so that I don’t have to specify all of them.

```
# Removing the labelled data type and making them factors instead
data_df_long <- data_df_long %>% unlabelled()
# Code to check the conversion
  #glimpse(data_df_long)
```

Most of the items are now factors, while the compound scores are still doubles. From what I can see, it looks like the conversion went well. The number of missing values are still the same and the factors seem corrrect. However, if something is acting weird down the road, this could be a likely source of the problem, since the function does several things at once (some of which might not be apparent now).

In addition, we need the wave variable to be numeric and not string. When ordering wave as a string, it will order it as 1, 10, 2, while numbers will be ordered as 1, 2, 10, which is what we want.

```
# Converting the wave variable to numeric
data_df_long$Wave <- data_df_long$Wave %>% as.numeric()
```

In addition, the factor levels for the young adult items of the CBCL is slightly different than for the other raters. For example, 2 is worded as “somewhat or sometimes true” in the young adult version, but “Somewhat/sometimes” for the other raters. I will change the wording of the factor levels for young adults to be consistent with that of the other raters, so that they can more easily be combined later.

```
data_df_long <- data_df_long %>%
  mutate(across(
    starts_with("YASR"),
    # The .x stands for the current column that it is iterating through
    ~ fct_recode(.x,
                 "Not" = "not true",
                 "Somewhat/sometimes" = "somewhat or sometimes true",
                 "Very/often" = "very true or often true")
  ))
```

8.2 - Filling in missing ages

This part is not needed anymore, as we got the additional data as a separate file. However, it will be kept here for reference.

As noted earlier, there does not seem to be any age variables for waves 8, 10, and 13. We will see whether we can get a hold of them. However, in the meantime, we will input a value for these ages, corresponding to the age of the child at the previous wave + 1. Looking over the intervals between the other waves, it seems to range from 1 to 2 years and be different for each child. We will use 1 year, as that seems to be the most likely given the wave interval (even though some intervals would likely have been higher). This substitution is meant to only be temporary.

The following code confirms that we are lacking age data for waves 8, 10, and 13, in the data frame (based on the age variable):

```
# Printing a matrix with the count of the different ages per wave
table(round(data_df_long$Age, digits = 0), data_df_long$Wave)
```

The easiest way to do this in dplyr seems to be by referring to the previous data point, and making sure that is the previous data point by arranging the waves in ascending order. This requires that there are no missing waves in the data set, so I will make sure the number of data points for wave 7, 8, 9, 10, 12, and 13 are the same, if not, it will fail. In addition, the filling in based on order will not work if the Wave column has string objects, as they are ordered after first character (e.g. 1, 10, 2), so the objects have to be numerical. After checking those things, the ages can be filled in based on the value in the previous row.

```
# This code is a bit of a mess. Essentially, it is counting how many data points we have in the
data set per wave and if they are not all the same, you get an error
if (data_df_long %>% group_by(Wave) %>% count() %>% ungroup() %>% select(n) %>% unique() %>%
length() != 1) {
  stop(
    "The data frame inputed did not fail the error check to see whether there number of data points
per wave is the same. It should be the same based on the wide to long conversion, so if you get
this error, you might have messed up the data after the conversion or inputted the wrong data
frame. The following code depends on ordering by waves, so the ages will likely be estimated
wrongly if there aren't the same number of data points for each wave."
  )
}

# Check whether the wave data is numeric, if not, ordering will be off
if (!is.numeric(data_df_long$Wave)) {
  stop(
    "The code that calculates the ages for the waves that do not have that data relies on the wave
variable being numeric. If not, the ordering of the waves will be off, as it will be ordered based
on each character (e.g. 1, 13, 20). The following error has been called since the wave variable for
the current data frame is not numeric, meaning that the next code block cannot be calculated
properly."
  )
}

# The following code was used for checking that the code was run correctly. I am saving it here as
a comment for reference.
  #test <- data_df_long %>% filter(ID %in% c("200006", "200012"))
  #test %>% arrange(Wave) %>% select(Wave, Age, ID) %>% print(n = 25)

# Filling in the age variables at waves 8, 10, and 13 based on the age at  previous wave + 1. If
that value is NA, the new age will also be NA.
data_df_long <- data_df_long %>%
  arrange(Wave) %>%
  group_by(ID) %>%
  mutate(
    Age  = case_when(
    Wave == 8 ~ lag(Age) + 1,
    Wave == 10 ~ lag(Age) + 1,
    Wave == 13 ~ lag(Age) + 1,
    TRUE ~ Age
  )) %>%
  ungroup()
```

8.3 - Rounded age variable

We need a new variable that is the rounded age variable, as several of our data analyses will use age as a categorical variable.

```
data_df_long <- data_df_long %>%
  mutate(
    Age_int = round(Age, digits = 0)
  )
```

Note that the round() function in R will round 8.50 to 8, but 8.51 to age 9. This is according to IEEE standards. This should however not have much of a practical significance as the ages are reported to 6 decimals, leaving it highly unlikely that any age will be x.500000.

8.4 - Age categories variable

Some of the analyses will compare the values based on the age. This requires that there are enough data points per group to be representative. Therefore, we will add another age variable for the age category. We will keep all ages that have more than 300 data points per rater and per age its separate category, but combine ages that have fewer than that number of data points.

Check number of data points per age of child and rater combo for externalizing symptoms, filtering out missing values:

```
data_df_long %>%
    select(ext_scale_f, ext_scale_m, ext_scale_c, Age_int) %>%
    pivot_longer(cols = c(ext_scale_f, ext_scale_m, ext_scale_c), names_to = "rater", values_to =
"rating") %>%
    mutate(
        rater = case_when(
            rater == "ext_scale_f" ~ "Father",
            rater == "ext_scale_m" ~ "Mother",
            rater == "ext_scale_c" ~ "Child"
        )) %>%
    filter(!is.na(rating)) %>%
    group_by(rater, Age_int) %>% count()
```

```
# A tibble: 47 × 3
# Groups:   rater, Age_int [47]
   rater Age_int     n
   <chr>   <dbl> <int>
 1 Child       7    54
 2 Child       8   580
 3 Child       9  1014
 4 Child      10  1149
 5 Child      11   689
 6 Child      12   332
 7 Child      13   439
 8 Child      14   478
 9 Child      15   523
10 Child      16   689
# ℹ 37 more rows
```

Check number of data points per age of child and rater combo for internalizing symptoms, filtering out missing values:

```
data_df_long %>%
    select(int_scale_f, int_scale_m, int_scale_c, Age_int) %>%
    pivot_longer(cols = c(int_scale_f, int_scale_m, int_scale_c), names_to = "rater", values_to =
"rating") %>%
  mutate(
        rater = case_when(
            rater == "int_scale_f" ~ "Father",
            rater == "int_scale_m" ~ "Mother",
            rater == "int_scale_c" ~ "Child"
        )) %>%
  filter(!is.na(rating)) %>%
    group_by(rater, Age_int) %>% count()
```

```
# A tibble: 55 × 3
# Groups:   rater, Age_int [55]
   rater Age_int     n
   <chr>   <dbl> <int>
 1 Child       7    55
 2 Child       8   587
 3 Child       9  1019
 4 Child      10  1140
 5 Child      11   691
 6 Child      12   332
 7 Child      13   441
 8 Child      14   476
 9 Child      15   520
10 Child      16   683
# ℹ 45 more rows
```

We aim to have at least 200 data points per rater per age, so that means that we have to combine ages 7-8 into one category, as well as ages 18-25. We will keep this consistent over raters and symptoms to ease comparison. The differences were also rather minimal between raters and symptoms.

It looks like we need to combine age 7 and 8, as well as ages 23-25 into categories for the age_cat variable

```
data_df_long <- data_df_long %>%
  mutate(Age_cat = case_when(
    Age_int <= 8 ~ "7-8",
    Age_int >= 18 ~ "18-25",
    TRUE ~ as.character(Age_int)
    ))
```

Make the Age_cat variable a factor for easier handling later:

```
data_df_long$Age_cat <- data_df_long$Age_cat %>%
  as.factor() %>%
  factor(levels = c("7-8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18-25"))
```

8.5 - Drawing unique observations from age categories

With age categories, there will be several observations of the same age when in the age range for the category. For example, in the category 18-25, there could be multiple observations of the same child, such as from when they were 18, 19, 20, and 21, years old. Therefore, we will draw one random observation per child from the age categories including several ages, such as 7-8 and 18-25. In addition, there are also some waves that are shorter than a year apart, also leading to multiple observations per age group for the age groups with only one age in it.

```
# Setting a seed for reproducibility
set.seed(1814)

# Creating a new data frame, data_df_long_unique from choosing one random observation per child
from data_df_long
data_df_long <- data_df_long %>%
  # Group by ID to get one observation per child
  group_by(ID, Age_cat) %>%
  # Chose one random observation based on the grouping variable
  slice_sample() %>%
  # Ungroup the grouping by ID
  ungroup()
```

9 - Saving data_df_long as a Rds file

```
saveRDS(data_df_long, file = "data_df_long.rds")

# You can now retrieve the data with data_df_long <- readRDS(file = "data_df_long.rds")
```

10 - Long to Wide Conversion

There might be some scenarios in which it would be beneficial to have the data frame in wide format, as it was originally. However, the data_df_long data frame has all the edits from the data wrangling. Therefore, I will convert the data_df_long back to a data_df_wide format, but which is data wrangled.

Long to wide conversion:

```
data_df_wide <- data_df_long %>%
  pivot_wider(id_cols = c(ID,  IDN, m1educ_m, f1educ_f, ChGender), names_from = Wave, values_from =
setdiff(names(data_df_long), c("ID", "IDN", "m1educ_m", "f1educ_f", "ChGender", "Wave")))

# Changing the column names to have the same format as they originally had, where yx where x is a
number designates the wave number. The following code finds column names that have some symbols (in
a capture group) and ends with "_x" where x is a number in a capture group. It then adds the prefix
"y", then the number in the second capture group, then the original column name. The "_" is deleted
from the column name.
colnames(data_df_wide) <- gsub("^(.*)_(\\d+)$", "y\\2\\1", colnames(data_df_wide))
```

11 - Saving data_df_wide as a Rds file

```
saveRDS(data_df_wide, file = "data_df_wide.rds")
```

```
# You can now retrieve the data with data_df_wide <- readRDS(file = "data_df_wide.rds")
```

10 - Data set overview

- raw_df - has the raw data loaded in from the SPSS file “Data_raw.sav”
- data_df - has changed prefixed and suffixes for data handling and consistency. For example, the variables for the first wave now has the pre-fix y1 instead of nothing. Age variable names are standardized.
- data_df_long - wide to long conversion of the data. Also has calculated scale scores for externalizing and internalizing symptoms for CBCL variations and age rounded up. In addition to some changed variable types for columns. The data frame also has the doubles with labels columns converted to factors with unlabelled(). Lastly, we are missing the age variable at waves 8, 10, and 13, so the ages have been filled in at these waves with the age at the previous wave + 1.
- data_df_wide - long to wide conversion of the data_df_long data frame. It therefore has all the same changes as those of data_df_long, but in a wide format. The column names follow the same naming conventions as data_df with yxname, where y is a prefix, x designates the wave, and name is the name of the variable. Note that variables that weren’t recorded at a wave, will now have a column for that variable with all missing data. Therefore, one might instead want to filter by columns that do not have all missing values or look at data_df to see what variables were recorded at each wave.

11 - Be aware that:

- The question labels for the columns are based on the earlier version of the survey. Occasionally, there was small changes in the wording of the question as the child got older. These are not reflected in the column variable labels. Refer to the code book for that.
- The participants got the child or youth version of the CBCL questionnaire based on wave, not age. Therefore, there might be some ages where some participants answered the child and other the young adult questionnaire
- The int_scale_c and ext_scale_c has some different items depending on the wave the data was collected in. If it matters what exact wording was used, filter the scores based on whether the wave is < or > than 9. (see code for calculating the scale for more information). In addition, there were some slight differences between the wording in int_scale_m/f and ext_scale_m/f between waves 9???