# Methods 3 – Portfolio Assignment 1

Christian, Ane, Anne Christine & Hannah Mai

The objective of this warm-up assignment is to brush up your programming and data wrangling skills as well as to introduce you to the data set you will be working on in Assignment 1. More precisely, you will here be asked here to do the following:

1. Introduction to version control
   - Create a **Github** (or **gitlab**) account
   - Link your **Git** account to your **RStudio**
   - Create a new repository/project
2. Introduction to the data set
3. Data Manipulation refresher

- Transform several data sets into a single one
  - Filter data so as to keep only the variables needed for next assignments
  - Tidyverse packages (**stringr** and **dplyr**)

## A1.1: Introduction to version control

In the assignments you will be asked to upload your code on **Github** and the **GitHub** repositories will be part of the portfolio, therefore all students must make an account and link it to their **RStudio** (you'll thank us later for this!).

- Install latest **R** version
- Install latest **RStudio** version
- Create a Git account and link it to your RStudio

N.B. Create a GitHub repository for the Assignment 1 and link it to a project on your **RStudio**.

## A1.2: Introduction to the dataset

**Project:** Language development in Autism Spectrum Disorder (ASD)

**Source:** @fusaroli2019hearing

**Background:** Autism Spectrum Disorder (ASD) is often related to language impairment, and language impairment strongly affects the patients ability to function socially (maintaining a social network, thriving at work, etc.). It is therefore crucial to understand how language abilities develop in children with ASD, and which factors affect them (to figure out e.g. how a child will develop in the future and whether there is a need for language therapy).

However, language impairment is always assessed by relying on the parent, teacher or clinician's subjective judgment of the child, and measured very sparsely (e.g. at 3 years old and again at 6). To help address this gap in clinical practice, we videotaped around 30 kids with ASD and around 30 comparison kids (matched by linguistic performance at visit 1) for approximately 30 minutes of naturalistic interactions with a parent. We repeated the data collection 6 times per kid at an interval of 4 months between each visit.

We transcribed the data and counted:

1. the amount of words that each kid uses in each video. Same for the parent.
2. the amount of unique words that each kid uses in each video. Same for the parent.

3. the amount of morphemes per utterance (Mean Length of Utterance) displayed by each child in each video. Same for the parent.

Different data sets were built by the researchers involved in the project:

1) demographic and clinical data about the children (recorded by a clinical psychologist)
2) length of utterance data (calculated by a linguist)
3) amount of unique and total words used

Your job in this assignment is to double-check the data and make it analysis-ready for the next assignment, in which we will try to understand how children language development is a function of cognitive and social factors and how the latter can be used as "cues" to likely future language impairments.

## A1.3: Data Manipulation Refresher

If you have created a Rstudio project for this assignment, the working directory for this assignment (the default path for any function, procedure or command that works on files) is the project directory. If you have decided otherwise, first make sure that you set your working directory to the path of the current RMarkdown file. You can also use the following code chunk to install and/or load the packages you will need for the tasks below.

```
# echo = TRUE prints code (and results) to HTML output file.
knitr::opts_chunk$set(echo = TRUE)

#set working directory.
knitr::opts_knit$set(root.dir = '~/Documents/GitHub/CAHA')
#getwd()

# clear environment.
rm(list=ls()) # clears global workspace.


#Loading in packages
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(stringr)
```

Load the following three data sets, after downloading them from dropbox and saving them in your working directory:

- Demographic data of the participants
- Length of utterance data
- Word data

```
#Loading in data
Demo_participants <- read.csv("demo_train.csv")
```

```
lu_data <- read.csv("LU_train.csv")
Word_data <- read.csv("token_train.csv")
```

Using various visualisation and aggregation procedures, explore these data sets and try to compare them to each other and draw parallel. You'll quickly notice that this is not an easy task. This is in part due to the fact that the different data sets were built by different researchers at different points in time. In particular, you'll find out that:

- the same variables might have different names (e.g. participant and visit identifiers)
- the same variables might report the values in different ways (e.g. participant and visit IDs)

Given this, it is important to make sure that all the relevant variables are identical in both name, type, and possible values.

**1. Renaming variables**

The first task thus consists in identifying which variable names are spelled differently and rename them accordingly.

**Tip:** To find the right procedures and functions, you can:

- look through the chapter on data transformation in the book *R for data science* [@wickham2023]
- familiarize yourself with the package **dplyr** (which is part of the tidyverse)
- google "how to rename variables in R"
- check the **janitor** R package.

Many different procedures can be used here, and unless you have to deal with gigantic data sets and have severe limitations in terms of computation time and space, there is no need for optimization here. Whatever works works.

```
#Renaming "Visit" variable
lu_data <- rename(lu_data, Visit = VISIT)
Word_data <- rename(Word_data, Visit = VISIT)

#Rename "ID" variable
Demo_participants <- rename(Demo_participants, ID = Child.ID)
lu_data <- rename(lu_data, ID = SUBJ)
Word_data <- rename(Word_data, ID = SUBJ)
```

**2. Renaming values**

Find a way to homogenize the way "visit" is reported amongst the different datasets. The function **str_extract()** from the package **stringr** can help you here: with the use of the regular expression '//d', this function allows you to capture the first occurrence of a number (*d* is for *digit*) within a string.

```
#Making the "Visit" value numerical
lu_data$Visit <- str_extract(lu_data$Visit, "\\d")
Word_data$Visit <- str_extract(Word_data$Visit, "\\d")
#?str_extract
```

A similar task needs to be done regarding the value names of the Child.ID variable in the demographic data set. The values of this variable that are not abbreviated do not end with "." (i.e. Adam), whereas they do in the other two data sets (i.e. Adam.). Key merges, that is, merging of data sets based on shared variables, can only be done is the latter have overlapping value names; if no identical value names can be found, nothing will be merged. In the present case, a simple way to ensure this consists in removing all points from the values of the corresponding variables. The package **stringr** can again be of help here, notably the function **str_replace_all()**.

**Tip:** You can either have one line of code for each child name that needs to be changed (easier, more typing) or specify the pattern that you want to replace (more complicated: look up "regular expressions", but more

generic and thus less typing)

```
### Removing the dots after the names in the ID column
Demo_participants$ID <- str_replace_all(Demo_participants$ID, "\\.$", "")
lu_data$ID <- str_replace_all(lu_data$ID, "\\.$","")
Word_data$ID <- str_replace_all(Word_data$ID, "\\.$","")
```

### 4. Data subsetting

This task consists in transforming the three data sets so as to keep only the variables that are of relevance for the project. For this purpose, the function **select()** from the **tidyverse** package **dplyr** will do nicely.

The variables of relevance here are the following:

- Child.ID,
- Visit,
- Diagnosis,
- Ethnicity,
- Gender,
- Age,
- ADOS,

- MullenRaw,
- ExpressiveLangRaw,
- Socialization
- MOT_MLU,
- CHI_MLU,
- types_MOT,
- types_CHI,
- tokens_MOT,
- tokens_CHI.

Most variables names should make sense, that is, should be straightforward as regards to the information it contains. Here are the less intuitive ones among the ones listed above:

- *ADOS* (Autism Diagnostic Observation Schedule) indicates the severity of the autistic symptoms (the higher the score, the worse the symptoms). Source: @lord2000autism
- *MLU* stands for 'Mean Length of Utterance'
- *types* stands for unique words. For example, the same word appearing multiple times only accounts for 1 word type.
- *tokens* stands for overall amount of words. Each occurrence of any word is a token. This example will help better understand the type/token distinction: in the sentence "the horse is a horse, of course, of course", there are 9 tokens (9 words in all), but only 6 types (the, horse, is, a, of, course).
- *MullenRaw* indicates non verbal IQ, as measured by the Mullen Scales of Early Learning (MSEL)@lee2013mullen
- *ExpressiveLangRaw* indicates verbal IQ, as measured by MSEL
- *Socialization* indicates social interaction skills and social responsiveness, as measured by @volkmar1987social

Feel free to rename the variables into something you think is more intuitive (i.e. *nonVerbalIQ*, *verbalIQ*)

```
#Renaming Columns
Demo_participants <- rename(Demo_participants, nonverbalIQ = MullenRaw)
Demo_participants <- rename(Demo_participants, verbalIQ = ExpressiveLangRaw)

#Selecting relevant variables:
Demo_participants <- Demo_participants %>%
  select(
```

```
    ID,
    Visit,
    Diagnosis,
    Ethnicity,
    Gender,
    Age,
    ADOS,
    nonverbalIQ,
    verbalIQ,
    Socialization)

lu_data <- lu_data %>%
  select(
    Visit,
    ID,
    MOT_MLU,
    CHI_MLU)


Word_data <- Word_data %>%
  select(
    Visit,
    ID,
    types_MOT,
    types_CHI,
    tokens_MOT,
    tokens_CHI
  )
```

**5. Data merge**

Following completion of the previous cleaning procedures, the different data sets can now be merged into a single one.

It is important here to check if merging the data sets:

- has resulted in any loss of relevant data
- has resulted in the creation of NAs within the merged data set. If this is so, it is important to understand why these NAs were created (e.g. some measures were not taken at all visits, some recordings were lost or permission to use was withdrawn).

```
mergeddf1 <- merge(Demo_participants, lu_data, by = c("ID", "Visit"), all = T)
merged_dataset <- merge(mergeddf1, Word_data, by = c("ID", "Visit"), all = T)



# Check for any loss of relevant data
missing_data <- setdiff(union(names(Demo_participants), union(names(lu_data), names(Word_data))), names
if (length(missing_data) > 0) {
  cat("The following relevant variables were lost during merging:\n")
  cat(paste(missing_data, collapse = ", "), "\n")
} else {
  cat("No relevant data was lost during merging.\n")
}
```

```
## No relevant data was lost during merging.
```

```
# Check for NAs in the merged data set
nas <- sum(is.na(merged_dataset))

if (nas > 0) {
  cat("Total NAs in the merged data set:", nas, "\n")
} else {
  cat("No NAs were created in the merged data set.\n")
}
```

```
## Total NAs in the merged data set: 819
```

```
colSums(is.na(merged_dataset))
```

```
##            ID         Visit     Diagnosis     Ethnicity        Gender
##             0             0             0             0             0
##           Age          ADOS   nonverbalIQ      verbalIQ Socialization
##            10           247           190           249             3
##       MOT_MLU       CHI_MLU     types_MOT     types_CHI    tokens_MOT
##            20            20            20            20            20
##    tokens_CHI
##            20
```

**6. Data filtering**

In order for our models to be useful, we want to minimize the need to actually test children as they develop. In other words, we would like to be able to predict the children's linguistic development after only having tested them once. Therefore we need to make sure that our *ADOS*, *MullenRaw*, *ExpressiveLangRaw* and *Socialization* variables are reporting (for all visits) only the scores from visit 1.

A possible way to do so:

- create a new data set having only first visits as rows as well as *child.ID* and the 4 relevant clinical variables as columns
- rename the clinical variables in a way that clearly indicates they relate to the first visit (e.g. *ADOS* to *ADOS1*)
- merge the new data set with the old

```
#Creating new df with the five columns from 1st visit
df_visit1 <- merged_dataset %>%
    filter(Visit == "1") %>%
    select(ID, ADOS, nonverbalIQ, verbalIQ, Socialization, Visit)

# Rename the clinical variables to indicate they relate to the first visit
first_visit_data <- df_visit1 %>%
  rename(ADOS1 = ADOS,
         nonverbalIQ1 = nonverbalIQ,
         verbalIQ1 = verbalIQ,
         Socialization1 = Socialization)

Cleaned_data_merged <- merge(merged_dataset, first_visit_data, by = c("ID"), all = T)

#Removing "Visit.y" and renaming "Visit.x"

Cleaned_data_merged <- rename(Cleaned_data_merged, Visit = Visit.x)
Cleaned_data_merged <- subset(Cleaned_data_merged, select = -Visit.y )
```

**7. Reverse Encoding**

An important part of data cleaning is making sure both variable names and value allow for easy and intuitive analysis and interpretation. For example, the different values for the categorical variable *gender* have been encoded to '1' and '2', but these numbers are useless to anyone who doesn't have the proper encoding dictionary, which maps each number to the corresponding category, namely 'male' or 'female'. In the following code chunk, resolve this ambiguity by reversing the encoding of the gender variable, so that 1 and 2 are changed to F and M respectively. In the same vein, transform the 'Diagnosis' variable so that 'A's and 'B's are reverted back to ASD (Autism Spectrum Disorder) and TD (typically developing) respectively.

```
# Reverse the encoding of 'gender' variable
Cleaned_data_merged <- Cleaned_data_merged %>%
  mutate(Gender = recode(Gender, "1" = "M", "2" = "F"))

# Reverse the encoding of 'Diagnosis' variable
Cleaned_data_merged <- Cleaned_data_merged %>%
  mutate(Diagnosis = recode(Diagnosis, "A" = "ASD", "B" = "TD"))
```

### 8. Saving cleaned data

Finally, save the cleaned data in Comma-Separated Values (CSV) format, under a name and at a location of your own choosing.

```
# Choose a file path and name where you want to save the CSV file
file_path <- "~/Documents/GitHub/CAHA/Cleaned_data_merged"

# Save the cleaned data to a CSV file
write.csv(Cleaned_data_merged, file = file_path, row.names = FALSE)
```

Our data is now ready for the next assignment!