DVEP Data Analysis

Gustavo Santos Paiva Laender Moura

2025-01-05

# READ ME

## Exporting data from REDCap

1. PID 1958
2. Data Exports, Reports, and Stats
3. Choose “Export Data” from option “A - All data (all records and fields)”
4. Export format: CSV (raw data)
5. De-identification options: “Remove all identifier fields” YES
6. Additional export options: “Export survey identifier field and survey timestamp field(s)?” YES
7. Advanced data formatting options
   * “Export gray Form Status fields with blank value”
   * comma as default separator
   * Use period/full stop (.) as decimal
8. Export file to “~Data Science/PROJECTS/DVEP”
9. Rename file to “data\_dvep.csv” Note: resulting file must have 780 variables (last variable on column ACZ: “anexos\_complete”)

## Files

### Codebooks and supporting files

* **codebook\_dvep.xlsx**: Codebook for REDCap DVEP Project
* **codebook\_structure.csv:** Data structure for DVEP project
* **codebook\_ncit.csv** Operational Support - NCIT coding
* **codebook\_bia.xlsx** Codebook for BIA data

### Data

* **data\_dvep.csv** Full data from REDCap DVEP Project.
* **data\_bia\_D1.csv** BIA data for the first visit for all participants. Contains BIA data from the first visit only. Contains BIA data from participants who did not complete the intervention.
* **data\_bia\_D3.csv** BIA data for participants that completed the intervention. Contains BIA data from both the first and third visits.

# DATA WRANGLING

## Getting started with R

# 1. Getting started with R  
## Clear existing data and graphics  
rm(list = ls())  
graphics.off()  
cat("\014") # Clear any pending RStudio sessions or temporary files  
  
## Load necessary libraries  
library(tidyverse)  
library(readxl)  
library(lubridate)  
library(stringr)  
library(purrr)  
library(gt)  
library(jmv)  
  
  
## Set working directory  
setwd("/Users/gustavosplmoura/Library/Mobile Documents/com~apple~CloudDocs/Medicina/Biblioteca/Research/Data Science/Data Science/PROJECTS/DVEP")

## Read CSV data files (Tidyverse)

# 2. Read CSV data files (Tidyverse)  
codebook\_dvep <- read\_excel(  
 "codebook\_dvep.xlsx",  
 col\_names = TRUE,  
 col\_types = NULL,  
 na = c("", "NA", "NI", "UNK", "NASK", "ASKU", "INV"),  
 trim\_ws = TRUE,  
 skip = 0, # Number of lines to skip before reading data  
 n\_max = Inf, # Maximum number of lines to read.  
 guess\_max = 1000  
 ) |>  
 arrange(index)  
  
codebook\_bia <- read\_excel(  
 "codebook\_bia.xlsx",  
 col\_names = TRUE,  
 col\_types = NULL,  
 na = c("", "NA", "NI", "UNK", "NASK", "ASKU", "INV"),  
 trim\_ws = TRUE,  
 skip = 0, # Number of lines to skip before reading data  
 n\_max = Inf, # Maximum number of lines to read.  
 guess\_max = 1000  
 ) |>  
 arrange(index)  
  
codebook\_structure <- read\_csv(  
 "codebook\_structure.csv",  
 col\_names = TRUE) |>   
 select(  
 form\_name\_en:V3  
 )  
  
codebook\_ncit <- read\_csv(  
 "codebook\_ncit.csv",  
 col\_names = TRUE)  
  
data <- read\_csv(  
 "data\_dvep.csv",  
 col\_names = TRUE,  
 col\_types = NULL,  
 col\_select = NULL,  
 id = NULL,  
 locale = default\_locale(),  
 na = c("", "NA", "NI", "UNK", "NASK", "ASKU", "INV"),  
 quote = "\"",  
 comment = "",  
 trim\_ws = TRUE,  
 skip = 0, # Number of lines to skip before reading data  
 n\_max = Inf, # Maximum number of lines to read.  
 guess\_max = 1000,  
 name\_repair = "unique",  
 num\_threads = readr\_threads(),  
 progress = show\_progress(),  
 show\_col\_types = TRUE,  
 skip\_empty\_rows = TRUE,  
 lazy = should\_read\_lazy()  
)  
  
data\_bia\_D3 <- read\_csv(  
 "data\_bia\_D3.csv",  
 col\_names = TRUE)  
  
data\_bia\_D1 <- read\_csv(  
 "data\_bia\_D1.csv",  
 col\_names = TRUE)

## Remove identifying data from record\_id

# 3. Remove identifying data from record\_id  
data$record\_id <- substr(data$record\_id,1,2)  
data\_bia\_D3 $File <- substr(data\_bia\_D3 $File,1,2)  
data\_bia\_D1$File <- substr(data\_bia\_D1$File,1,2)

## Renaming variables

# 4. Renaming variables  
# 4.1 data  
rename\_data <- setNames(object = colnames(data), codebook\_dvep$variable)  
data <- data |>  
 rename(!!!rename\_data)  
  
rm(rename\_data)  
  
# 4.2 bia  
rename\_bia <- setNames(object = colnames(data\_bia\_D3 ), codebook\_bia$variable)  
  
data\_bia\_D3 <- data\_bia\_D3 |>  
 rename(!!!rename\_bia)  
  
data\_bia\_D1 <- data\_bia\_D1 |>  
 rename(!!!rename\_bia)  
  
rm(rename\_bia)

## record\_id as.integer

# 5. record\_id as.integer  
data$record\_id <- as.integer(data$record\_id)  
data\_bia\_D3 $record\_id <- as.integer(data\_bia\_D3 $record\_id)  
data\_bia\_D1$record\_id <- as.integer(data\_bia\_D1$record\_id)

## Assign labels to variables

# 6. Assign labels to variables with base R attr()  
data <- data |>   
 mutate(across(  
 all\_of(codebook\_dvep$variable),  
 ~ {  
 attr(., "label") <- codebook\_dvep$label\_pt[codebook\_dvep$variable == cur\_column()]  
 .  
 }  
 ))  
  
data\_bia\_D3 <- data\_bia\_D3 |>   
 mutate(across(  
 all\_of(codebook\_bia$variable),  
 ~ {  
 attr(., "label") <- codebook\_bia$label\_pt[codebook\_bia$variable == cur\_column()]  
 .  
 }  
 ))

## Creating functions

#### filter\_variables()

filter\_variables(visit = c("eleg", "V1", "V2", "V3"), include\_repeating = NULL, form\_name = NULL, filter\_included = TRUE)

Arguments:

* visit: A vector of visit names to filter (default: all visits)
* include\_repeating = NULL
  + If NULL, includes all variables
  + If 0, does not include repeating variables
  + If 1, only shows repeating variables
* form\_name = NULL: The name of the form to filter on (NULL means no filtering by form)
* filter\_included = TRUE: Whether to filter by the ‘included’ column (default: TRUE)

filter\_variables <- function(  
 visit = c("eleg", "V1", "V2", "V3"),   
 include\_repeating = NULL,   
 form\_name = NULL,   
 filter\_included = TRUE   
 ) {  
   
 # Validate the 'visit' input: Ensure all provided visit names are valid  
 if (!all(visit %in% c("eleg", "V1", "V2", "V3"))) {  
 stop("Invalid visit name. Choose from 'eleg', 'V1', 'V2', or 'V3'.")  
 }  
   
 # Define valid form names for validation  
 valid\_form\_names <- c(  
 "eleg", "demographic", "whoqol", "dass", "ecap", "measures", "bp\_limb", "bp",   
 "bia", "handgrip", "eliminations", "evs", "alcohol", "tobacco", "diet\_recall",   
 "intake", "dates", "allocation", "conditions", "drugs", "old.drugs", "history",   
 "symptoms", "phy.exam", "labs", "ecg", "compliance", "events", "medical",   
 "followup", "conclusion"  
 )  
   
 # Validate the 'form\_name' input: Ensure it contains only valid form names  
 if (!is.null(form\_name) && !all(form\_name %in% valid\_form\_names)) {  
 stop("Invalid form\_name. Choose from: ", paste(valid\_form\_names, collapse = ", "))  
 }  
   
 # Filter the 'codebook\_dvep' based on the specified criteria  
 filtered\_codebook <- codebook\_dvep |>   
 filter(  
 # If filter\_included is TRUE, filter for rows where 'included' equals 1  
 if (filter\_included) included == 1 else TRUE,  
   
 # Retain rows where at least one of the selected visits has a value greater than 0  
 rowSums(across(all\_of(visit))) > 0,  
   
 # If 'include\_repeating' is specified, filter by the repeating\_instrument column  
 if (!is.null(include\_repeating)) repeating\_instrument == include\_repeating else TRUE,  
   
 # If 'form\_name' is specified, filter by the form\_name\_en column  
 if (!is.null(form\_name)) form\_name\_en == form\_name else TRUE  
 )  
   
 # Extract and return the 'variable' column from the filtered codebook  
 filtered\_vars <- filtered\_codebook$variable  
   
 return(filtered\_vars) # Return the filtered variable names  
}

#### filter\_data()

filter\_data <- function( visit = c("eleg", "V1", "V2", "V3"), include\_repeating = NULL, form\_name = NULL)

Arguments:

* visit: A vector of visit names to filter (default: all visits)
* include\_repeating = NULL
  + If NULL, includes all variables
  + If 0, does not include repeating variables
  + If 1, only shows repeating variables
* form\_name = NULL: The name of the form to filter on. **Will only work for repeating instruments.**

## 7.2 filter\_data() based on visit, repeating instrument and form  
filter\_data <- function(  
 visit = c("eleg", "V1", "V2", "V3"),   
 include\_repeating = NULL,   
 form\_name = NULL) {  
   
 # Map visit names to actual event\_name values  
 mapped\_visits <- case\_when(  
 visit == "eleg" ~ "eleg\_arm\_1",  
 visit == "V1" ~ "1visit\_arm\_1",  
 visit == "V2" ~ "2visit\_arm\_1",  
 visit == "V3" ~ "3visit\_arm\_1",  
 TRUE ~ visit  
 )  
   
 # Map form\_name to repeat\_instrument values using a case\_when structure  
 mapped\_form\_name <- if (!is.null(form\_name)) {  
 case\_when(  
 form\_name == "eleg" ~ "elegibilidade",  
 form\_name == "demographic" ~ "dados\_demogrficos",  
 form\_name == "whoqol" ~ "questionrio\_qualidade\_de\_vida",  
 form\_name == "dass" ~ "escore\_de\_depresso\_ansiedade\_e\_estresse",  
 form\_name == "ecap" ~ "escala\_de\_compulso\_alimentar",  
 form\_name == "measures" ~ "antropometria",  
 form\_name == "bp\_limb" ~ "presso\_arterial\_determinao\_do\_membro\_de\_referncia",  
 form\_name == "bp" ~ "presso\_arterial",  
 form\_name == "bia" ~ "impedncia\_bioeltrica\_corporal",  
 form\_name == "handgrip" ~ "fora\_de\_preenso\_palmar",  
 form\_name == "eliminations" ~ "avaliao\_nutricional",  
 # form\_name == "allergies" ~ "alergia\_alimentar",  
 form\_name == "evs" ~ "exercise\_vital\_sign",  
 form\_name == "alcohol" ~ "consumo\_alcool",  
 form\_name == "tobacco" ~ "consumo\_tabaco",  
 form\_name == "diet\_recall" ~ "recordatrio\_alimentar",  
 form\_name == "intake" ~ "avaliao\_da\_ingesto\_alimentar",  
 form\_name == "dates" ~ "datas\_importantes",  
 form\_name == "allocation" ~ "nmero\_do\_participante",  
 form\_name == "conditions" ~ "comorbidades",  
 form\_name == "drugs" ~ "medicamentos\_de\_uso\_habitual",  
 form\_name == "old.drugs" ~ "medicamentos\_prvios",  
 form\_name == "history" ~ "antecedentes\_pessoais",  
 form\_name == "symptoms" ~ "sintomas",  
 form\_name == "phy.exam" ~ "exame\_fsico",  
 form\_name == "labs" ~ "exames\_laboratoriais",  
 form\_name == "ecg" ~ "eletrocardiograma",  
 form\_name == "compliance" ~ "adeso",  
 form\_name == "events" ~ "eventos\_adversos",  
 form\_name == "medical" ~ "avaliao\_mdica",  
 form\_name == "followup" ~ "contato\_semanal",  
 form\_name == "conclusion" ~ "concluso",  
 # form\_name == "annex" ~ "anexos",  
 TRUE ~ form\_name  
 )  
 } else {  
 NULL  
 }  
   
 # Get the filtered variable names using the filter\_variables function  
 filtered\_vars <- filter\_variables(visit, include\_repeating, form\_name)  
   
 # Filter the raw data to only include these columns and match event\_name and form\_name  
 filtered\_data <- data %>%   
 filter(event\_name %in% mapped\_visits) %>%  
 filter(if (!is.null(include\_repeating) && include\_repeating == 0) is.na(repeat\_instrument) | repeat\_instrument == "" else TRUE) %>%  
 filter(if (!is.null(include\_repeating) && include\_repeating == 1) !is.na(repeat\_instrument) & repeat\_instrument != "" else TRUE) %>%  
 filter(if (!is.null(mapped\_form\_name)) repeat\_instrument == mapped\_form\_name else TRUE) %>%  
 select(record\_id, event\_name, repeat\_instrument, repeat\_instance, all\_of(filtered\_vars)) %>%  
 mutate(  
 repeat\_instrument = ifelse(is.na(repeat\_instrument), "", repeat\_instrument),  
 repeat\_instance = ifelse(repeat\_instrument == "", NA, repeat\_instance)  
 )  
   
 return(filtered\_data)  
}

#### filter\_codebook()

filter\_codebook(form\_name = c(...), included = 1)

Arguments

* form\_name
* included = 1: defaults to 1, filtering variables by included column. If set to 0, will include all variables

## 7.3 filter\_codebook()  
filter\_codebook <- function(form\_name = c(  
 "eleg", "tcle", "demographic", "whoqol", "dass", "ecap", "measures",   
 "bp\_limb", "bp", "bia", "handgrip", "eliminations", "allergies",   
 "evs", "alcohol", "tobacco", "diet\_recall", "intake", "dates",   
 "allocation", "conditions", "drugs", "old.drugs", "history",   
 "symptoms", "phy.exam", "labs", "ecg", "compliance", "events",   
 "medical", "followup", "conclusion", "annex"),  
 included = 1) {  
   
 # Ensure input is valid  
 if (!all(form\_name %in% c("eleg", "tcle", "demographic", "whoqol", "dass", "ecap", "measures",   
 "bp\_limb", "bp", "bia", "handgrip", "eliminations", "allergies",   
 "evs", "alcohol", "tobacco", "diet\_recall", "intake", "dates",   
 "allocation", "conditions", "drugs", "old.drugs", "history",   
 "symptoms", "phy.exam", "labs", "ecg", "compliance", "events",   
 "medical", "followup", "conclusion", "annex")))   
 {  
 stop("Invalid form name")  
 }  
   
 if (included == 1) {  
 codebook\_form <- codebook\_dvep |>   
 filter(form\_name\_en %in% form\_name & included == 1)  
 } else {  
 codebook\_form <- codebook\_dvep |>   
 filter(form\_name\_en %in% form\_name)  
 }  
  
 return(codebook\_form)  
}

#### convert\_col\_type()

convert\_col\_type(data, codebook = codebook\_dvep)

Arguments:

* data: dataframe to apply the function
* codebook = codebook\_dvep: codebook source. Defaults to codebook\_dvep

Tips:

* as.factor(): categorical data where the label (e.g., “6 cápsulas ao dia”) is more meaningful than numeric code.
* binary data (0, Não | 1, Sim):
  + Use as.factor() if the “label” (Não or Sim) is important.
  + Use as.numeric(as.character()) if you’re performing mathematical operations (e.g., calculating proportions, averages).
* For ordinal data (1, Ruim | 2, Regular | 3, Boa | 4, Excelente): use as.factor() with ordered levels (ordered()) if you need to preserve the ranking.

## 7.4 convert\_col\_type()  
convert\_col\_type <- function(data, codebook = codebook\_dvep) {  
 # Nested function to convert a single column  
 convert\_column <- function(column, type) {  
 switch(type,  
 f = as.factor(column), # Factor  
 o = as.factor(column), # Factor  
 c = as.character(column), # Character  
 d = as.numeric(column), # Numeric  
 i = as.integer(column), # Integer  
 k = lubridate::ymd(column), # Date (YYYY-MM-DD)  
 t = lubridate::ymd\_hms(column), # Date-Time (YYYY-MM-DD HH:MM:SS)  
 h = lubridate::hms(column), # Time only (HH:MM:SS)  
 n = as.numeric(as.character(column)), # Coerce to Numeric  
 l = as.logical(column), # Logical  
 D = as.Date(column, format = "%Y-%m-%d"), # Date with specified format  
 T = as.POSIXct(column, format = "%Y-%m-%d %H:%M:%S"), # Date-Time  
 column # Default (no change)  
 )  
 }  
 # Apply conversion  
 data <- data |>  
 mutate(  
 across(  
 .cols = all\_of(intersect(colnames(data), codebook$variable)), # Ensures only common variables are processed  
 .fns = ~ convert\_column(.x, codebook$col\_types[which(codebook$variable == cur\_column())])  
 )  
 )  
}  
convert\_col\_type(data)

#### label\_variables()

label\_variables(data, codebook, language = "pt")

Arguments:

* data: dataframe to which function will be applied
* codebook: source codebook
* language = "pt": defaults to portuguese ("pt"); set "en" for english

label\_variables <- function(data, codebook, language = "pt") {  
 # Determine the label column based on the language argument  
 label\_column <- ifelse(language == "en", "label\_en", "label\_pt")  
  
 # Ensure column names are consistent  
 codebook\_vars <- codebook$variable  
 codebook\_labels <- codebook[[label\_column]]  
  
 # Identify common variables in both data and codebook  
 common\_vars <- intersect(names(data), codebook\_vars)  
  
 # Loop through the common variables and assign labels  
 for (var in common\_vars) {  
 label <- codebook\_labels[codebook\_vars == var]  
 attr(data[[var]], "label") <- label  
 }  
  
 return(data)  
}

label\_variables2()

#### label\_choices()

label\_choices(data, codebook = codebook\_dvep)

Arguments:

* data: dataframe to which function will be applied
* codebook = codebook\_dvep: source codebook (dafaults to codebook\_dvep)

## 7.5 label\_choices()  
label\_choices <- function(data, codebook = codebook\_dvep) {  
 # Ensure required libraries are loaded  
 if (!requireNamespace("dplyr") || !requireNamespace("tidyr") ||  
 !requireNamespace("stringr") || !requireNamespace("purrr")) {  
 stop("Required libraries: dplyr, tidyr, stringr, purrr")  
 }  
   
 # 1. Filter codebook for relevant variables with col\_types in "f" or "o"  
 selected\_codebook <- codebook |>   
 filter(  
 variable %in% colnames(data) & # Variables present in data  
 col\_types %in% c("f", "o") # col\_types in "f" or "o"  
 )  
   
 # 2. Parse the `choices` column  
 parsed\_choices <- selected\_codebook |>  
 rowwise() |>   
 mutate(  
 parsed = list(  
 str\_split(choices, " \\| ") |> # Split choices by "|"  
 unlist() |>  
 map(~ str\_split\_fixed(.x, ", ", 2) |> # Split by ", " into two columns  
 as\_tibble(.name\_repair = "unique") |> # Ensure unique column names  
 setNames(c("raw\_value", "label")) # Name columns  
 ) |>  
 bind\_rows() # Combine into a tibble  
 )  
 ) |>  
 select(variable, parsed) |>  
 unnest(parsed) # Expand parsed choices into rows  
   
 # 3. Create lookup tables for selected variables  
 lookup\_tables <- parsed\_choices |>  
 group\_by(variable) |>  
 summarize(  
 lookup = list(setNames(label, raw\_value)), .groups = "drop"  
 ) |>  
 deframe()  
   
 # 4. Replace raw values with labels in data, using "Unmatched" for unmatched values  
 for (column\_name in names(lookup\_tables)) {  
 if (column\_name %in% colnames(data)) { # Ensure column exists in data  
 data[[column\_name]] <- recode( # Apply recode using the lookup table  
 data[[column\_name]],  
 !!!lookup\_tables[[column\_name]],  
 .default = "Unmatched" # Set "Unmatched" as the placeholder for unmatched values  
 )  
 }  
 }  
   
 # Explicitly return the modified data  
 return(data)  
}

## Bioimpedance data

#### Data from first/third visit

Applies to participants who completed the intervention

# 8. Wrangling Bioimpedance data  
## 8.1 D3 DATA (contains data from first and third visits for participants who completed the intervention)  
### Filter lines for which phaseangle is <> ""  
data\_bia\_D3\_filtered <- data\_bia\_D3 |>   
 filter(!is.na(phaseangle)) |>   
 mutate(  
 date = as.Date(timestamp), # Extract the date  
 time = format(timestamp, "%H:%M:%S") # Extract the time  
 ) |>   
 select(all\_of(codebook\_bia$variable)[codebook\_bia$included == 1], date, time) |>   
 relocate(record\_id, date, time, phaseangle, raverage, xcaverage, weight:w\_ecwbytbw) |>   
 arrange(record\_id, date, time)  
  
### Group by record\_id and date and obtain mean of multiple measurements from the same day  
data\_bia\_D3\_filtered <- data\_bia\_D3\_filtered |>   
 group\_by(record\_id, date) |>  
 summarise(  
 across(c(phaseangle:m\_tohimaginary), \(x) mean(x, na.rm = TRUE)),  
 .groups = "drop"  
 ) |>  
 group\_by(record\_id) |> # Add coding for visit number  
 mutate(  
 visit = case\_when(  
 date == min(date) ~ 1, # Assign 1 to the earliest date  
 date == max(date) ~ 3, # Assign 3 to the latest date  
 TRUE ~ NA\_real\_ # Default to NA for unexpected cases  
 ),  
 .after = record\_id  
 )

#### Data from first visit

Applies to participants who did not complete the intervention

## 8.2 D1 DATA (data from the first visit for participants who did not complete the intervention)  
## Filter lines for which phaseangle is <> ""  
data\_bia\_D1\_filtered <- data\_bia\_D1 |>   
 filter(!is.na(phaseangle)) |>   
 mutate(  
 date = as.Date(timestamp), # Extract the date  
 time = format(timestamp, "%H:%M:%S") # Extract the time  
 ) |>   
 select(all\_of(codebook\_bia$variable)[codebook\_bia$included == 1], date, time) |>   
 relocate(record\_id, date, time, phaseangle, raverage, xcaverage, weight:w\_ecwbytbw) |>   
 arrange(record\_id, date, time)  
  
### Group by record\_id and date and obtain mean of multiple measurements from the same day  
data\_bia\_D1\_filtered <- data\_bia\_D1\_filtered |>   
 group\_by(record\_id, date) |>  
 summarise(  
 across(c(phaseangle:m\_tohimaginary), \(x) mean(x, na.rm = TRUE)),  
 .groups = "drop"  
 ) |>  
 group\_by(record\_id) |> # Add coding for visit number  
 mutate(  
 visit = case\_when(  
 date == min(date) ~ 1, # Assign 1 to the earliest date  
 date == max(date) ~ 3, # Assign 3 to the latest date  
 TRUE ~ NA\_real\_ # Default to NA for unexpected cases  
 ),  
 .after = record\_id  
 )  
### Selecting BIA data from D1 not present in D3  
data\_bia\_D1\_filtered <- data\_bia\_D1\_filtered |>   
 filter(  
 record\_id %in% setdiff(1:75, data\_bia\_D3\_filtered$record\_id)   
 )

#### Merging to single tibble

## 8.3 MERGE D1/D3 BIA data into a single tibble  
data\_bia <- bind\_rows(  
 data\_bia\_D1\_filtered, data\_bia\_D3\_filtered  
) |>   
 mutate(  
 visit = as.integer(visit)  
 ) |>   
 arrange(  
 record\_id, visit  
 ) |>   
 ungroup()  
  
### label\_variables  
data\_bia <- label\_variables(data\_bia, codebook\_bia)

#### Drop intermediate tibbles

# 8.4. DROP intermediate tibbles  
rm(data\_bia\_D1)  
rm(data\_bia\_D1\_filtered)  
rm(data\_bia\_D3)  
rm(data\_bia\_D3\_filtered)

## Wrangling DVEP REDCap data

#### Adding NCIT labels

## 9.2 Repeating instruments  
### 9.2.1. Concatenating NCIT labels (from codebook\_ncit) to NCIT codes  
#### Conditions (commorbidities)  
I21\_conditions\_R <- filter\_data("eleg",1,"conditions") |>   
 left\_join(  
 codebook\_ncit |> select(ncit\_code, descriptive),  
 # by = c("common\_comorbidities" = "ncit\_code")  
 join\_by(common\_comorbidities == ncit\_code)  
 ) |>  
 relocate(  
 descriptive, .after = common\_comorbidities  
 )  
  
I21\_conditions\_R <- label\_variables(I21\_conditions\_R, codebook\_dvep)  
  
#### Drugs in regular use  
I22\_drugs\_R <- filter\_data("eleg",1,"drugs") |>   
 left\_join(  
 codebook\_ncit |> select(ncit\_code, descriptive),  
 join\_by(drugs\_sql == ncit\_code)  
 ) |>  
 relocate(  
 descriptive, .after = drugs\_sql  
 )  
  
I22\_drugs\_R <- label\_variables(I22\_drugs\_R, codebook\_dvep)  
  
# # 1.3 Previous drugs  
# I23\_old\_drugs\_R <- filter\_data("eleg",1,"old.drugs") |>   
# left\_join(  
# codebook\_ncit |> select(ncit\_code, descriptive),  
# join\_by(common\_previous\_medications == ncit\_code)  
# )|>  
# relocate(  
# descriptive, .after = common\_previous\_medications  
# )  
#   
# # 1.4 Past medical conditions  
# I24\_old\_conditions\_R <- filter\_data("eleg",1,"history") |>   
# left\_join(  
# codebook\_ncit |> select(ncit\_code, descriptive),  
# join\_by(common\_medical\_history == ncit\_code)  
# )|>  
# relocate(  
# descriptive, .after = common\_medical\_history  
# )  
  
  
# Most common comorbidities  
# I21\_conditions\_R |>   
# group\_by(common\_comorbidities, descriptive) |>   
# count(common\_comorbidities, sort = TRUE, name = "frequency") |>   
# mutate(percentage = round((frequency/75 \* 100),1)) |>   
# view()  
  
# NCIT Condition   
# C3117 Hipertensão 18 24 \*1  
# C26696 Ansiedade 16 21.3   
# C37967 Hipercolesterolemia 16 21.3 \*2  
# C37971 Hipertrigliceridemia 13 17.3 \*2  
# C113101 Resistência à insulina 11 14.7 \*3  
# C26800 Hipotireoidismo 9 12   
# C89715 Enxaqueca 8 10.7  
# C114667 SOP 7 9.3  
# C26747 DM2 7 9.3 \*3  
   
# Most common drugs  
# ```{r, eval = FALSE}  
# I22\_drugs\_R |>   
# group\_by(drugs\_sql, descriptive) |>   
# count(drugs\_sql, sort = TRUE, name = "frequency") |>   
# mutate(percentage = round((frequency/75 \* 100),1)) |>   
# view()

#### Exclusive variables from Eleg/D1 data\_d1\_exclusive

## 9.1 Eclusive variables from Eleg/D1 (`data\_d1\_exclusive`) to be replicated to d2 and d3  
eleg\_exclusive <- filter\_data("eleg",0) |>   
 mutate(  
 intervention\_duration = as.numeric(conclusion\_date - intervention\_start\_date)  
 )|>   
 select(record\_id, allocation\_group, completed\_intervention, intervention\_duration, non\_completion\_reason, age, sex)  
  
visit\_1\_exclusive <- filter\_data("V1",0) |>   
 select(  
 record\_id,  
 race:income\_level  
 ) |>   
# codebook\_dvep$choices[codebook\_dvep$variable == "race"]  
# [1] "c41260, Asiático | c41261, Branco origem europeia | c128994, Branco origem América do Sul | c16352, Negro | c17998, desconhecido | c17649, Outro"  
 mutate(  
 race = if\_else(race == "c41261", "c128994", race)  
 )  
  
data\_d1\_exclusive <- eleg\_exclusive |>   
 left\_join(  
 visit\_1\_exclusive,  
 by = join\_by(record\_id)  
)  
  
rm(eleg\_exclusive)  
rm(visit\_1\_exclusive)

### Repeating instruments

#### Creating relevant binary variables\*

Hypertension

### 9.3.1 HYPERTENSION  
#### Extract record IDs associated with hypertension diagnosis  
hypertension\_conditions <- I21\_conditions\_R |>   
 filter(str\_detect(common\_comorbidities, "C3117")) |>   
 pull(record\_id)  
  
#### Extract record IDs associated with antihypertensive drugs  
hypertension\_drugs <- I22\_drugs\_R |>   
 filter(str\_detect(drugs\_sql,   
 "C66869|C29098|C61635|C47640\_2|C28836|C29254|C62027|C62027\_2"  
 )  
 ) |>   
 pull(record\_id)  
  
#### Assign hypertension based on conditions or drugs  
data\_d1\_exclusive <- data\_d1\_exclusive |>   
 mutate(hypertension = if\_else(  
 record\_id %in% hypertension\_conditions,   
 1,   
 if\_else(  
 record\_id %in% hypertension\_drugs,   
 1,   
 0  
 )  
 ))  
  
rm(hypertension\_conditions)  
rm(hypertension\_drugs)

Dyslipidemia

### 9.3.2 DYSLIPIDEMIA  
#### Extract record IDs associated with dyslipidemia conditions  
dyslipidemia\_conditions <- I21\_conditions\_R |>   
 filter(str\_detect(common\_comorbidities, "C37967|C37971")) |>  
 pull(record\_id)  
  
#### Extract record IDs associated with antihypertensive drugs  
dyslipidemia\_drugs <- I22\_drugs\_R |>   
 filter(str\_detect(drugs\_sql,   
 "C29454|C66523\_2|C47529|C61527|C87471"  
 )  
 ) |>   
 pull(record\_id)  
  
#### Assign dyslipidemia based on conditions or drugs  
data\_d1\_exclusive <- data\_d1\_exclusive |>   
 mutate(dyslipidemia = if\_else(  
 record\_id %in% dyslipidemia\_conditions,   
 1,   
 if\_else(  
 record\_id %in% dyslipidemia\_drugs,   
 1,   
 0  
 )  
 ))  
  
rm(dyslipidemia\_conditions)  
rm(dyslipidemia\_drugs)

Insulin resistance

### 9.3.3 INSULIN RESISTANCE  
#### Extract record IDs associated with insulin resistance or diabetes  
insulin\_conditions <- I21\_conditions\_R |>   
 filter(str\_detect(common\_comorbidities, "C113101|C26747")) |>  
 pull(record\_id)  
  
#### Extract record IDs associated with anti-hyperglycemic / hypoglycemic drugs  
insulin\_drugs <- I22\_drugs\_R |>   
 filter(str\_detect(drugs\_sql,   
 "C61612|C61612\_2|C87618|C180533"  
 )  
 ) |>   
 pull(record\_id)  
  
#### Assign dyslipidemia based on conditions or drugs  
data\_d1\_exclusive <- data\_d1\_exclusive |>   
 mutate(insulin = if\_else(  
 record\_id %in% insulin\_conditions,   
 1,   
 if\_else(  
 record\_id %in% insulin\_drugs,   
 1,   
 0  
 )  
 ))  
  
rm(insulin\_conditions)  
rm(insulin\_drugs)

Drugs that might induce weight loss

### 9.3.4 DRUGS THAT MIGHT INDUCE WEIGHT LOSS  
#### Extract record IDs  
drugs\_w\_loss <- I22\_drugs\_R |>   
 filter(str\_detect(drugs\_sql,   
 "C61939|C62012|C506\_1|C1278\_2|C1278\_1|C1278\_3|C47764\_1|C47764\_2|C61680"  
 )  
 ) |>   
 pull(record\_id)  
  
#### Assign drugs\_w\_loss based on drugs  
data\_d1\_exclusive <- data\_d1\_exclusive |>   
 mutate(drugs\_w\_loss = if\_else(  
 record\_id %in% drugs\_w\_loss, 1, 0)  
 )  
  
rm(drugs\_w\_loss)

Drugs that might induce weight gain

### 9.3.5 DRUGS THAT MIGHT INDUCE WEIGHT GAIN  
#### Extract record IDs  
drugs\_w\_gain <- I22\_drugs\_R |>   
 filter(str\_detect(drugs\_sql,   
 "C61879|C62005|C61917\_2|C29416|C29536\_2"  
 )  
 ) |>   
 pull(record\_id)  
  
#### Assign drugs\_w\_loss based on drugs  
data\_d1\_exclusive <- data\_d1\_exclusive |>   
 mutate(drugs\_w\_gain = if\_else(  
 record\_id %in% drugs\_w\_gain, 1, 0)  
 )  
  
rm(drugs\_w\_gain)

\*Relevant binary variables:

* Hypertension present if: C3117 Hipertensão C66869 Losartana C29098 Hidroclorotiazida C61635 Anlodipino C47640\_2 Olmesartana C28836 Atenolol C29254 Metoprolol C62027 Enalapril 10 mg C62027\_2 Enalapril 20 mg
* Dyslipidemia present if: C37967 Hipercolesterolemia C37971 Hipertrigliceridemia C29454 Sinvastatina C66523\_2 Rosuvastatina C47529 Ezetimiba C61527 Atorvastatina C87471 Ciprofibrato
* Insulin resistance present if: C113101 Resistência insulínica C26747 DM2 C61612 Metformina 500 mg C61612\_2 Metformina 850 mg C87618 Gliclazida 30 mg C180533 Empagliflozin/Linagliptin
* Drugs that might induce weight loss C61939 Sertralina C62012 Bupropiona C506\_1 Fluoxetina C1278\_2 Venlafaxina 75 mg C1278\_1 Venlafaxina 37,5 mg C1278\_3 Venlafaxina 150 mg C47764\_1 Topiramato 25 mg C47764\_2 Topiramato 50 mg C61680 Citalopram 20 mg
* Drugs that might induce weight gain C61879 Paroxetina 20 mg C62005 Amitriptilina 25 mg C61917\_2 Quetiapina 50 mg C29416 Risperidona 2 mg C29536\_2 Ácido Valpróico 250 mg “C61879|C62005|C61917\_2|C29416|C29536\_2”

###### Wrapping up data\_d1\_exclusive

data\_d1\_exclusive <- label\_choices(data\_d1\_exclusive, codebook\_dvep)  
data\_d1\_exclusive <- convert\_col\_type(data\_d1\_exclusive, codebook\_dvep)  
data\_d1\_exclusive <- data\_d1\_exclusive |>   
 mutate(  
 hypertension = as.factor(hypertension),  
 dyslipidemia = as.factor(dyslipidemia),  
 insulin = as.factor(insulin),  
 drugs\_w\_loss = as.factor(drugs\_w\_loss),  
 drugs\_w\_gain = as.factor(drugs\_w\_gain)  
 )  
data\_d1\_exclusive <- label\_variables(data\_d1\_exclusive, codebook\_dvep)

#### Lab exames

I27\_labs\_R <- filter\_data(c("V1","V2","V3"),1,"labs") |>   
 mutate(  
 visit = case\_when(  
 event\_name == "1visit\_arm\_1" ~ 1,  
 event\_name == "2visit\_arm\_1" ~ 2,  
 event\_name == "3visit\_arm\_1" ~ 3  
 ),  
 .after = record\_id  
 )|>   
 select(-event\_name, -repeat\_instrument, -repeat\_instance, -labs\_checked\_results\_yn)

#### Compliance

compliance\_V2 <- data |>   
 select(  
 record\_id, event\_name,  
 filter\_variables("V2",1,"compliance")  
 ) |>   
 filter(event\_name == "2visit\_arm\_1" & cp\_compliance\_complete == 2) |>   
 left\_join(data |>  
 filter(event\_name == "eleg\_arm\_1" & !is.na(intervention\_start\_date)) |>   
 select(record\_id,intervention\_start\_date, conclusion\_date),  
 by = join\_by(record\_id)  
 ) |>   
 left\_join(data |>  
 filter(event\_name == "2visit\_arm\_1" & !is.na(evaluation\_date)) |>   
 select(record\_id,evaluation\_date),  
 by = join\_by(record\_id)  
 ) |>   
 rename(evaluation\_date\_2 = evaluation\_date)  
  
compliance\_V3 <- data |>   
 select(  
 record\_id, event\_name,  
 filter\_variables("V3",,"compliance")  
 ) |>   
 filter(event\_name == "3visit\_arm\_1" & cp\_compliance\_complete == 2) |>   
 left\_join(data |>  
 filter(event\_name == "eleg\_arm\_1" & !is.na(intervention\_start\_date)) |>   
 select(record\_id,intervention\_start\_date, conclusion\_date),  
 by = join\_by(record\_id)  
 ) |>   
 left\_join(data |>  
 filter(event\_name == "2visit\_arm\_1" & !is.na(evaluation\_date)) |>   
 select(record\_id,evaluation\_date),  
 by = join\_by(record\_id)  
 ) |>   
 rename(evaluation\_date\_2 = evaluation\_date)  
  
I29\_compliance <- bind\_rows(  
 compliance\_V2,compliance\_V3  
) |>   
 mutate(  
 record\_id = as.integer(record\_id)  
 ) |>   
 mutate(  
 visit = case\_when(  
 event\_name == "2visit\_arm\_1" ~ 2,  
 event\_name == "3visit\_arm\_1" ~ 3  
 ),  
 .after = record\_id  
 ) |>   
 arrange(record\_id,visit) |>   
 select(record\_id, visit, intervention\_start\_date, evaluation\_date\_2, conclusion\_date, cp\_taking\_as\_directed\_yn, cp\_schedule, cp\_schedule\_other, cp\_missed\_dose\_yn, cp\_missed\_dose\_count, cp\_discontinued\_yn, cp\_discontinued\_n\_days, cp\_discontinued\_reason\_other, cp\_ran\_out\_of\_drug\_yn, cp\_ran\_out\_reason, cp\_perceived\_improvement\_yn, cp\_perceived\_improvement, cp\_medication\_confidence\_scale, cp\_self\_reported\_compliance\_rate) |>   
 convert\_col\_type()  
  
rm(compliance\_V2)  
rm(compliance\_V3)  
  
I29\_compliance <- label\_variables(I29\_compliance, codebook\_dvep)  
I29\_compliance <- label\_choices(I29\_compliance, codebook\_dvep)

#### Adverse events

## 9.9 Adverse events  
I30\_events\_R <- filter\_data(,1,"events") |>   
 filter(  
 cp\_adverse\_event\_this\_cycle\_yn == 1  
 ) |>  
 mutate(  
 visit = case\_when(  
 event\_name == "1visit\_arm\_1" ~ 1,  
 event\_name == "2visit\_arm\_1" ~ 2,  
 event\_name == "3visit\_arm\_1" ~ 3  
 ),  
 .after = record\_id  
 )|>   
 select(-event\_name, -repeat\_instrument, -cp\_additional\_adverse\_events\_yn, -cp\_adverse\_event\_this\_cycle\_yn)  
  
I30\_events\_R <- label\_variables(I30\_events\_R, codebook\_dvep)  
I30\_events\_R <- label\_choices(I30\_events\_R, codebook\_dvep)

### Non-repeating instruments

#### Data common to V1 and V3 (d1d3)

### Non-repeating data common to V1 and V3 (`d1d3`)  
# calculate mean of handgrip strenght  
# select relevant variables  
d1d3 <- filter\_data(c("V1","V3"),0) |>   
 mutate(  
 handgrip = if\_else(  
 is.na(handgrip\_right\_mean) & is.na(handgrip\_left\_mean),  
 NA\_real\_, # Leave blank (NA) if both are missing  
 if\_else(  
 !is.na(handgrip\_right\_mean) & is.na(handgrip\_left\_mean),  
 handgrip\_right\_mean, # Use the right hand value if left is missing  
 if\_else(  
 is.na(handgrip\_right\_mean) & !is.na(handgrip\_left\_mean),  
 handgrip\_left\_mean, # Use the left hand value if right is missing  
 rowMeans(cbind(handgrip\_right\_mean, handgrip\_left\_mean), na.rm = TRUE) # Calculate mean if both are present  
 )  
 )  
 )  
 ) |>   
 mutate(  
 visit = case\_when(  
 event\_name == "1visit\_arm\_1" ~ 1,  
 event\_name == "3visit\_arm\_1" ~ 3  
 )  
 ) |>   
 select(  
 record\_id, visit,  
 whoqol\_score\_overall, # 4. whoqol  
 dass\_score\_depression:ecap\_score, # 5. dass, 6. ecap  
 height, weight, abdomen, arm, bmi, # 7. measures  
 mean\_bp\_mean, # 9. bp   
 time\_fasted\_food, time\_fasted\_liquid, resistance, reactance, phase\_angle, # 10. bia  
 handgrip, # 11. handgrip  
 # 12. eliminations  
 evs\_score, # 14. evs  
 alcohol\_dose, alcohol\_significant, # 15. alcohol  
 smoke\_history, pack\_years, # 16. tobacco  
 carbs\_kcal, protein\_kcal, fat\_kcal, # 18. intake  
 drugs\_dose\_change\_yn, drugs\_dose\_change\_notes, # 31. medical  
 intervention\_prevention\_reason\_yn, # 31. medical  
 specify\_intervention\_prevention\_reasons, # 31. medical  
 intervention\_delivered\_yn, # 31. medical  
 explain\_intervention\_not\_delivered # 31. medical  
 )

#### Data from the second visit (d2)

## 9.5 Non-repeating data from the second visit (`d2`)  
d2 <- filter\_data("V2",0) |>   
 mutate(  
 visit = case\_when(  
 event\_name == "2visit\_arm\_1" ~ 2  
 )  
 ) |>   
 select(  
 record\_id, visit,  
 height, weight, abdomen, arm, bmi, # 7. measures  
 mean\_bp\_mean, # 9. bp   
 # 12. eliminations  
 evs\_score, # 14. evs  
 drugs\_dose\_change\_yn, drugs\_dose\_change\_notes, # 31. medical  
 intervention\_prevention\_reason\_yn, # 31. medical  
 specify\_intervention\_prevention\_reasons, # 31. medical  
 intervention\_delivered\_yn, # 31. medical  
 explain\_intervention\_not\_delivered # 31. medical  
 )

### Joining

#### Bind rows from d1d3 and d2: data\_filtered

## 9.6 Bind rows for non-repeating variables from D1/D2/D3  
data\_filtered <- bind\_rows(  
 d1d3,d2  
) |>   
 mutate(  
 record\_id = as.integer(record\_id),  
 visit = as.integer(visit)  
) |>   
 arrange(  
 record\_id, visit  
 ) |>   
 convert\_col\_type()  
  
rm(d1d3)  
rm(d2)

#### Left\_joins

##### BIA data to data\_filtered

## 9.7. Left\_join BIA data to `data\_filtered`  
data\_filtered <- data\_filtered |>   
 left\_join(  
 data\_bia |>   
 select(  
 record\_id, visit,  
 phaseangle, raverage, xcaverage,   
 weight, height, waist, pal, bmi,   
 fmi, ffmi, vat,  
 w\_tbw, w\_ecw  
 ),  
 by = join\_by(record\_id, visit)  
 )

##### Compliance data to data\_filtered

data\_filtered <- data\_filtered |>   
 left\_join(  
 I29\_compliance,  
 by = join\_by(record\_id, visit)  
 )

##### data\_d1\_exclusive to all data

data\_filtered <- data\_d1\_exclusive |>   
 right\_join(  
 data\_filtered,  
 by = join\_by(record\_id)  
 ) |>   
 relocate(  
 visit,  
 .after = record\_id  
 )

##### Labs to data\_filtered

data\_filtered <- data\_filtered |>   
 left\_join(I27\_labs\_R,  
 by = join\_by(record\_id, visit)  
 )  
  
data\_filtered <- label\_variables(data\_filtered, codebook\_dvep)  
data\_filtered <- label\_variables(data\_filtered, codebook\_bia)  
data\_filtered <- data\_filtered |>   
 mutate(  
 visit = as.integer(visit)  
 )

### Supertible (data\_instruments)

Creates a supertibble with one tibble for each instrument

form\_names <- unique(codebook\_dvep$form\_name\_en)  
form\_names <- form\_names[-2]  
  
# Dynamically create the instruments list  
instruments <- setNames(  
 lapply(form\_names, function(form\_name) {  
 filter\_codebook(form\_name, 0)$variable  
 }),  
 paste0("I", sprintf("%02d", seq\_along(form\_names)), "\_", form\_names)  
)  
  
# Estas variáveis não devem ser consideradas na verificação de dados faltantes (NA) porque sempre contêm informações.  
always\_present\_vars <- c("record\_id", "event\_name", "repeat\_instrument", "repeat\_instance")  
  
# Criar uma lista de tibbles separadas para cada instrumento, excluindo linhas que contenham apenas NAs  
data\_instruments <- lapply(names(instruments), function(instr\_name) {  
   
 # `instr\_name` é o nome atual do instrumento sendo processado, por exemplo, "elegibility".  
   
 # Seleciona a lista de variáveis associadas ao instrumento atual  
 selected\_vars <- instruments[[instr\_name]]  
   
 # Remove as variáveis da lista que estão em `always\_present\_vars` (que sempre possuem valores).  
 # `setdiff()` retorna apenas as variáveis exclusivas (aquelas que não estão em `always\_present\_vars`).  
 vars\_to\_check <- setdiff(selected\_vars, always\_present\_vars)  
   
 # Filtrar os dados para o instrumento atual  
 filtered\_tibble <- data |>   
 # Seleciona as colunas correspondentes às variáveis do instrumento atual  
 select(all\_of(selected\_vars)) |>   
   
 # Filtra as linhas onde pelo menos uma das variáveis relevantes (não constantes) não é NA  
 filter(  
 rowSums(  
 !is.na(  
 select(cur\_data(), all\_of(vars\_to\_check)) # Seleciona apenas as colunas relevantes para a verificação de NA  
 )  
 ) > 0 # `rowSums()` conta quantas colunas não são NA por linha. Mantemos linhas onde este total é maior que 0.  
 )  
   
 # Retorna a tibble filtrada com as variáveis e linhas relevantes para o instrumento atual  
 return(filtered\_tibble)  
})  
  
# Nomeia os elementos da lista `data\_instruments` com os nomes correspondentes dos instrumentos.  
# Por exemplo, o primeiro elemento da lista será nomeado "redcap", o segundo "elegibility", e assim por diante.  
names(data\_instruments) <- names(instruments)  
  
rm(always\_present\_vars)  
rm(form\_names)  
rm(instruments)  
  
# Opcional: Salvar cada tibble no ambiente global como um objeto independente.  
# `list2env()` converte cada elemento da lista `data\_instruments` em um objeto no ambiente global,  
# com o nome correspondente ao instrumento.  
#list2env(data\_instruments, .GlobalEnv)

### Exports tibbles to Generated data folder

output\_dir <- '/Users/gustavosplmoura/Library/Mobile Documents/com~apple~CloudDocs/Medicina/Biblioteca/Research/Data Science/Data Science/PROJECTS/DVEP/Generated data'  
  
  
# Iterate over `data\_instruments`  
for (instr\_name in names(data\_instruments)) {  
 # Create the file path for the current instrument  
 file\_path <- file.path(output\_dir, paste0(instr\_name, ".csv"))  
   
 # Write the tibble to a CSV file  
 write\_csv(data\_instruments[[instr\_name]], file\_path)  
   
 # Print a message confirming the export  
 message("Exported: ", file\_path)  
}  
  
# Additional tibbles  
write\_csv(data\_bia, file.path(output\_dir, "data\_bia.csv"))  
write\_csv(data\_d1\_exclusive, file.path(output\_dir, "data\_d1\_exclusive.csv"))  
write\_csv(data\_filtered, file.path(output\_dir, "data\_filtered.csv"))  
  
rm(output\_dir)  
rm(file\_path)  
rm(instr\_name)

# DATA ANALYSIS

## Simplifying Environment

codebooks <- tibble(  
 name = c("bia", "dvep", "ncit", "structure"),  
 data = list(codebook\_bia, codebook\_dvep, codebook\_ncit, codebook\_structure)  
)  
  
# Assign names to the `data` list-column  
names(codebooks$data) <- codebooks$name  
  
# Remove the individual tibbles from the environment  
rm(codebook\_bia, codebook\_dvep, codebook\_ncit, codebook\_structure)

### Pull individual codebooks

You can pull each individual codebook by:

* codebooks$data[[“bia”]]
* codebooks$data[[“dvep”]]
* codebooks$data[[“ncit”]]
* codebooks$data[[“structure”]]

## Functions

### summarize\_numerical()

summarize\_numerical(data, group\_col = NULL, use\_labels = TRUE)

Arguments: data: dataframe group\_col = NULL: grouping column; defaults to NULL use\_labels = TRUE: If set to true, uses labels instead of variable names

summarize\_numerical <- function(data, group\_col = NULL, use\_labels = TRUE) {  
 # Extract variable labels, falling back to variable names if labels are missing  
 variable\_labels <- sapply(names(data), function(var) {  
 label <- attr(data[[var]], "label")  
 if (is.null(label) || !use\_labels) var else label  
 }, simplify = TRUE)  
   
 # Identify numeric columns  
 numeric\_cols <- data %>% select(where(is.numeric)) %>% names()  
   
 if (!is.null(group\_col)) {  
 # Grouped summary  
 summary <- data %>%  
 group\_by(across(all\_of(group\_col))) %>%  
 summarise(across(  
 all\_of(numeric\_cols),  
 ~ paste0(  
 round(mean(.x, na.rm = TRUE), 1),   
 " (",   
 round(mean(.x, na.rm = TRUE) - 1.96 \* sd(.x, na.rm = TRUE) / sqrt(sum(!is.na(.x))), 1),   
 "–",   
 round(mean(.x, na.rm = TRUE) + 1.96 \* sd(.x, na.rm = TRUE) / sqrt(sum(!is.na(.x))), 1),   
 ")"  
 ),  
 .names = "{.col}"  
 )) %>%  
 pivot\_longer(-all\_of(group\_col), names\_to = "Variable", values\_to = "Value") %>%  
 mutate(Variable = variable\_labels[Variable]) # Replace variable names with labels/names  
 } else {  
 # Ungrouped summary  
 summary <- data %>%  
 summarise(across(  
 all\_of(numeric\_cols),  
 ~ paste0(  
 round(mean(.x, na.rm = TRUE), 1),   
 " (",   
 round(mean(.x, na.rm = TRUE) - 1.96 \* sd(.x, na.rm = TRUE) / sqrt(sum(!is.na(.x))), 1),   
 "–",   
 round(mean(.x, na.rm = TRUE) + 1.96 \* sd(.x, na.rm = TRUE) / sqrt(sum(!is.na(.x))), 1),   
 ")"  
 ),  
 .names = "{.col}"  
 )) %>%  
 pivot\_longer(everything(), names\_to = "Variable", values\_to = "Value") %>%  
 mutate(Variable = variable\_labels[Variable]) # Replace variable names with labels/names  
 }  
   
 return(summary)  
}

### summarize\_categorical()

summarize\_categorical(data, group\_col = NULL, use\_labels = TRUE)

Arguments: - data: dataframe - group\_col = NULL: grouping column; defaults to NULL - use\_labels = TRUE: If set to true, uses labels instead of variable names

summarize\_categorical <- function(data, group\_col = NULL, use\_labels = TRUE) {  
 # Extract variable labels, falling back to variable names if labels are missing  
 variable\_labels <- sapply(names(data), function(var) {  
 label <- attr(data[[var]], "label")  
 if (is.null(label) || !use\_labels) var else label  
 }, simplify = TRUE)  
   
 # Identify categorical columns  
 categorical\_cols <- data %>% select(where(~ is.character(.x) || is.factor(.x))) %>% names()  
   
 if (!is.null(group\_col)) {  
 # For grouped data  
 summary <- lapply(categorical\_cols, function(col) {  
 data %>%  
 group\_by(across(all\_of(group\_col)), .drop = FALSE) %>%  
 count(!!sym(col), .drop = FALSE, name = "Freq") %>%  
 mutate(  
 Percent = round(100 \* Freq / sum(Freq), 1),  
 Variable = variable\_labels[col] # Use labels or names  
 ) %>%  
 rename(Level = !!sym(col)) %>%  
 ungroup()  
 }) %>%  
 bind\_rows()  
 } else {  
 # For ungrouped data  
 summary <- lapply(categorical\_cols, function(col) {  
 data %>%  
 count(!!sym(col), .drop = FALSE, name = "Freq") %>%  
 mutate(  
 Percent = round(100 \* Freq / sum(Freq), 1),  
 Variable = variable\_labels[col] # Use labels or names  
 ) %>%  
 rename(Level = !!sym(col))  
 }) %>%  
 bind\_rows()  
 }  
   
 # Arrange columns for consistency  
 summary <- summary %>%  
 select(Variable, Level, Freq, Percent, everything())  
   
 return(summary)  
}

### compare\_groups()

compare\_groups <- function(data, group\_col = "allocation\_group", use\_labels = TRUE) {  
   
 # Extract variable labels, default to variable names if labels are missing  
 # If use\_labels = TRUE, fetch the 'label' attribute for each variable.   
 # If no label exists or use\_labels = FALSE, default to the variable name.  
 variable\_labels <- sapply(names(data), function(var) {  
 label <- attr(data[[var]], "label") # Check for the 'label' attribute  
 if (is.null(label) || !use\_labels) var else label # Use variable name if label is missing  
 }, simplify = TRUE)  
   
 # Prepare results storage  
 # Create an empty data.frame to store test results with predefined columns  
 results <- data.frame(  
 Variable = character(), # The variable being tested  
 Test = character(), # The type of test (t-test or chi-squared/Fisher's)  
 Statistic = numeric(), # The test statistic value  
 P\_value = numeric(), # The p-value from the test  
 stringsAsFactors = FALSE # Ensure strings are not converted to factors  
 )  
   
 # Loop through all columns in the data, excluding the grouping column  
 for (var in setdiff(names(data), group\_col)) {  
 if (is.numeric(data[[var]])) {  
 # Run t-test for numeric variables  
 # Compares the means of the variable between the groups in group\_col  
 test\_result <- t.test(data[[var]] ~ data[[group\_col]])  
   
 # Append the t-test results to the results data.frame  
 results <- rbind(results, data.frame(  
 Variable = variable\_labels[var], # Use label or name  
 Test = "t-test", # Specify the test type  
 Statistic = round(test\_result$statistic, 2), # Round the t-statistic  
 P\_value = round(test\_result$p.value, 4) # Round the p-value  
 ))  
 } else if (is.factor(data[[var]]) || is.character(data[[var]])) {  
 # Run chi-squared test for categorical variables  
 # Create a contingency table for the variable and the group\_col  
 contingency\_table <- table(data[[var]], data[[group\_col]])  
   
 # Check expected counts to decide between chi-squared and Fisher's test  
 if (any(chisq.test(contingency\_table)$expected < 5)) {  
 # Use Fisher's exact test when expected counts are too small  
 test\_result <- fisher.test(contingency\_table)  
   
 # Append the Fisher's exact test results to the results data.frame  
 results <- rbind(results, data.frame(  
 Variable = variable\_labels[var], # Use label or name  
 Test = "Fisher's exact test", # Specify the test type  
 Statistic = NA, # No statistic for Fisher's test  
 P\_value = round(test\_result$p.value, 4) # Round the p-value  
 ))  
 } else {  
 # Use chi-squared test when expected counts are sufficient  
 test\_result <- tryCatch(  
 chisq.test(contingency\_table), # Perform the chi-squared test  
 error = function(e) list(statistic = NA, p.value = NA) # Handle potential errors  
 )  
   
 # Append the chi-squared test results to the results data.frame  
 results <- rbind(results, data.frame(  
 Variable = variable\_labels[var], # Use label or name  
 Test = "Chi-squared test", # Specify the test type  
 Statistic = round(test\_result$statistic, 2), # Round the chi-squared statistic  
 P\_value = round(test\_result$p.value, 4) # Round the p-value  
 ))  
 }  
 }  
 }  
   
 # Return results as a gt table  
 # Convert the results data.frame into a gt table for better visualization  
 gt\_table <- results %>%  
 gt() %>%  
 tab\_header(  
 title = "Hypothesis Test Results", # Main title for the table  
 subtitle = paste("Comparison of", group\_col) # Subtitle indicating the grouping variable  
 ) %>%  
 cols\_label(  
 Variable = "Variable", # Rename the Variable column  
 Test = "Test Type", # Rename the Test column  
 Statistic = "Test Statistic", # Rename the Statistic column  
 P\_value = "P-value" # Rename the P\_value column  
 )  
   
 return(gt\_table) # Return the formatted gt table  
}

1. **Extract Variable Labels**:
   * Fetches the label attribute for each variable.
   * If no label exists or if use\_labels = FALSE, defaults to the variable name.
2. **Prepare Results Storage**:
   * An empty data.frame is created to store test results, including the variable name/label, test type, test statistic, and p-value.
3. **Loop Through Variables**:
   * **Numeric Variables**:
     + A two-sample t-test is run to compare means between groups.
     + Test results (t-statistic and p-value) are appended to the results data.frame.
   * **Categorical Variables**:
     + A contingency table is created for the variable and the grouping column.
     + If expected counts in the table are too small, Fisher’s exact test is used.
     + Otherwise, a chi-squared test is performed.
     + Results are appended to the results data.frame.
4. **Error Handling**:
   * The chi-squared test may fail for certain edge cases (e.g., empty levels). Errors are caught and handled gracefully by returning NA for the statistic and p-value.
5. **Output as a gt Table**:
   * The results data.frame is converted to a nicely formatted gt table with a title and custom column labels.

Example Usage

# Run the function and display results  
test\_results <- compare\_groups(data\_d1\_exclusive)  
  
# Print the results  
print(test\_results)