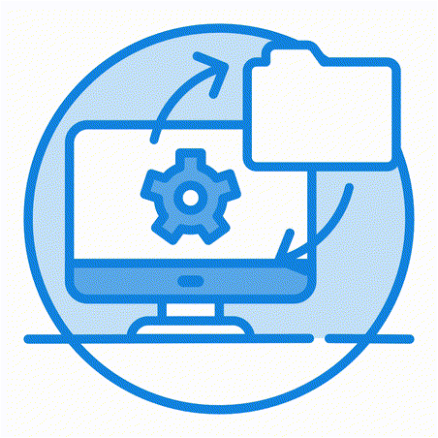
Swiss Paraplegic Research

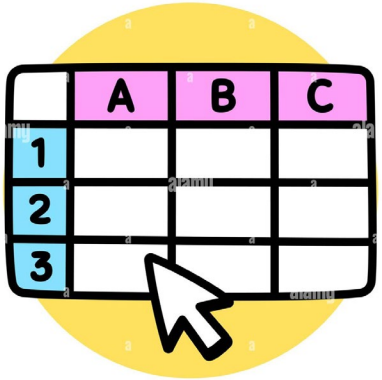
Workshop – Introduction into R

Data transformation using Base-R



Andreas Limacher

Data frame



- A data frame in R is a two-dimensional tabular data structure
- It consists of columns (variables) and rows (observations)
- Each column can contain data of different types, such as numeric, character, or logical
- All columns must have the same length, making it similar to a table or spreadsheet
- It can be thought of as a list of vectors of equal length
- Data frames are particularly useful for storing and manipulating structured data in R

Dataset

- Discuss dataset

```
> install.packages("NHANES")
> library(NHANES)
> data(NHANES)
```

- What kind of data does the dataset contain?
- How was the data collected?

NHANES

- The National Health and Nutrition Examination Survey (NHANES) is a survey that measures the health and nutrition of adults and children in the United States.
- The survey combines interviews, physical examinations and laboratory tests.
- The NHANES interview includes demographic, socioeconomic, dietary, and health-related questions. The examination component consists of medical, dental, and physiological measurements.
- Each year, about 5,000 adults and children in communities across the United States participate in NHANES. A random, scientific process to select the people is used.

Review data

- Review dataset

```
> class(NHANES)
> names(NHANES)
> head(NHANES)
> str(NHANES)
> summary(NHANES)
```

- How many variables and entries are there?
- What variable formats do exist?

Variable names

- As variable names are used very often, they must have clear syntax. We suggest the following:
  - Short names
  - No spaces (replace with underscores _)
  - No unusual characters (&, #, <, >, …)
  - Similar style nomenclature (e.g. all date variables named like date_onset, date_report, date_death…)

Select variables and entries

- Select single and multiple variables (columns) and entries (rows)

```
> weight <- NHANES$Weight
> weight <- NHANES[, "Weight"]
> weight <- NHANES[, 4]

> NHANES_subset <- NHANES[1:200, ]
> NHANES_subset <- NHANES_subset[, c("ID", "SurveyYr", "Gender", "Age")]
> NHANES_subset <- NHANES[1:300, c("ID", "SurveyYr", "Gender", "Age")]
```

Deduplication

- Identify and remove duplicates

```
> sum(duplicated(NHANES))
> NHANES_unique <- NHANES[!duplicated(NHANES), ]
```

- What issues can lead to duplicates?

Re-name and label variables

- Rename the variable

```
> names(NHANES_subset)[names(NHANES_subset) == "SurveyYr"] <-
  "Year"
```

- Label the variable

```
> attr(NHANES_subset$Year, "label") <- "Year of survey"
```

Generate new variable

- Calculate BMI

```
> NHANES$BMI_new <- NHANES$Weight / ((NHANES$Height/100)^2)
```

- Calculate high income

```
> summary(NHANES$HHIncomeMid)
> NHANES$HighIncome <- ifelse(NHANES$HHIncomeMid > 75000, 1, 0)
```

Convert variable format

- What variable format types exist in R?

Convert variable format

- What variable format types exist in R?
  - Numeric -> as.numeric()
  - Integer -> as.integer()
  - Character -> as.character()
  - Logical -> as.logical()
  - Factor -> as.factor()
  - Date -> as.Date()

Convert variable format

- R provides various functions for format conversion, allowing users to transform data between different types and structures.

```
> NHANES_subset$ID <- as.character(NHANES_subset$ID)
> NHANES_subset$Height <- as.integer(NHANES_subset$Height)
> NHANES_subset$temp <- as.factor(NHANES_subset$HighIncome)
> NHANES_subset$HighIncome <-
  factor(ifelse(NHANES_subset$HighIncome == 1, "Yes", "No"))
```

Factors

- In R, factors are used to work with categorical variables, variables that have a fixed and known set of possible values.
- Typically, a character or numeric variable is converted to a factor if you want to set an intrinsic order to the values (“levels”) so they can be displayed non-alphabetically in plots and tables.

```
> levels(NHANES$Gender)
> str(NHANES$Gender)
> summary(NHANES$Gender)
```

Re-order and re-label factor variables

- re-order

```
> NHANES$Gender <- factor(NHANES$Gender, levels = c("male",
  "female"))
> summary(NHANES$Gender)
```

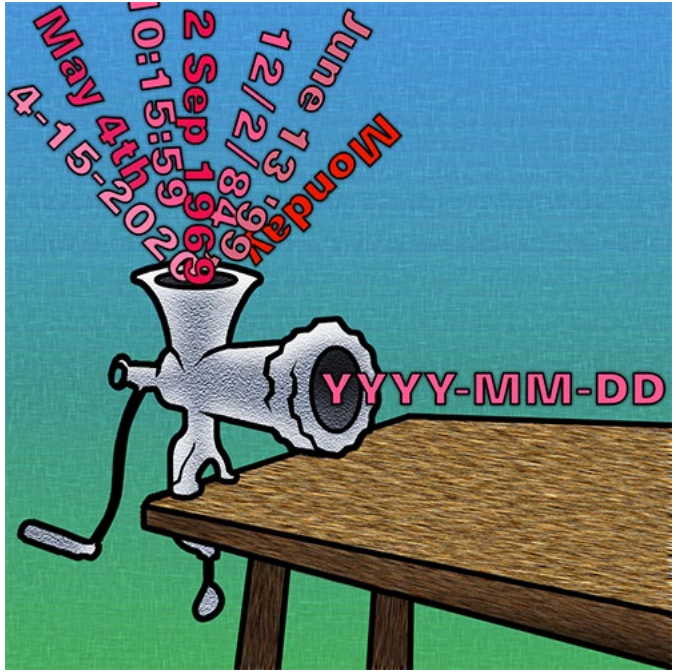
- re-label

```
> levels(NHANES$Gender) <- c("M", "F")
```

- add a category 'diverse’

```
> NHANES$Gender <- factor(NHANES$Gender, levels =
  c(levels(NHANES$Gender), "D"))
```

Date and time



Date and time

- Dates can be complicated to work with
  - Does every year have 365 days?
  - Does every day have 24 hours?
  - Does every minute have 60 seconds?
  - What is the reference time for world-wide studies?
- Different formats

```
> Sys.Date()
> Sys.time()
```

Dates

- It is important to make R recognize when a column contains dates.
- Dates are an object class and can be tricky to work with.
- After importing a dataset into R, date column values may look like “1989/12/30”, “05/06/2014”, or “13 Jan 2020”.
- R is likely treating these values as Character values. R must be told that these values are dates and what the format of the date is (which part is Day, which is Month, which is Year, etc).
- R converts these values to class Date. In the background, R will store the dates as numbers (the number of days from its “origin” date 1 Jan 1970).
- This allows to treat dates as continuous variables and to allow special operations such as calculating the difference between dates.
- By default, values of class Date are displayed as YYYY-MM-DD.

Convert string to date

- as.Date() is the standard base R function to convert an object or column to class Date
- Use the format = argument to tell R the current format of the character date components
- For example, if your character dates are currently in the format “DD/MM/YYYY”, like “24/04/1968”, then you would use format = "%d/%m/%Y"

```
> NHANES$SurveyYr <- as.character(NHANES$SurveyYr)
> NHANES$Survey_date <- paste0(NHANES$SurveyYr, "-01")
> NHANES$Survey_date <- as.Date(NHANES$Survey_date,
  format="%Y_%m-%d")
```

Iterate - lapply

- lapply is a function that allows you to apply a specific operation to each element of a list or vector
- For example, it can be applied to each column/variable of a data frame
- Syntax: lapply(X, FUN)

```
> result <- lapply(NHANES[,c("Age", "Weight", "Pulse")],
  function(x) mean(x, na.rm = TRUE))
> result
> par(mfrow = c(1, 3))
> lapply(NHANES[,c("Age", "Weight", "Pulse")], hist)
```

Iterate - for loop

- Helps you repeat a set of instructions multiple times
- Starts with a list of items you want to go through, like numbers or words
- Syntax:

```
for (item in list_of_items) {
    # Do something with each item
}
```

```
> for (number in 1:5) {
      print(number)
  }
> variables <- c("Age", "Weight", "Pulse")
> for (var in variables) {
      result[[var]] <- mean(NHANES[[var]], na.rm = TRUE)
  }
> result
```

Iterate over groups - by

- A useful tool for applying a function to subsets of data
- Syntax: by(data, grouping_factor, function_to_apply)

```
> by(NHANES$Age, NHANES$Gender, mean)
> by(NHANES$Weight, NHANES$Gender, mean)
> by(NHANES$Weight, NHANES$Gender, function(x) mean(x, na.rm =
  TRUE))
```

- Task: Calculate median age for each education level

Aggregate data

- A function for summarizing data based on groups and combining results into a new data frame.
- Syntax: aggregate(data_to_summarize, list(grouping_variable), function_to_apply)

```
> df <- aggregate(NHANES$Age, list(NHANES$Gender), mean)
> df
```

- Alternative: aggregate(variable_to_summarize ~ list(grouping_variable),data = dataset, function_to_apply)

```
> df <- aggregate(Age ~ Gender, data = NHANES, mean)
> df
```

if-else condition

- The if{} else{} statement allows for conditional execution of code
- Syntax:

```
if (condition) {
    # Code to execute if condition is true
} else {
   # Code to execute if condition is false
}
```

```
> result <- lapply(NHANES[c("Age","Gender","Weight","Pulse")], function(x) {
      if (is.numeric(x) | is.integer(x)) {
           mean(x, na.rm = TRUE)
      } else {
           prop.table(table(x))
      }
  })
> result
```

Exercise