Swiss Paraplegic Research

Workshop – Introduction into R

Data manipulation using tidyverse

Andreas Limacher

Tidyverse

- The tidyverse is a collection of R packages designed for data science that share a common design philosophy and syntax.
- Some core packages:
  - ggplot2 (data visualization)
  - dplyr (data manipulation)
  - tidyr (data tidying)
  - tibble (modern data frames)
- Consistent syntax: Uses a readable code style with functions that can be chained together using the pipe operator (%>%)
  - Shortcut: Control+Shift+M

Pros and cons of dplyr versus base R

- Pros
  - dplyr is significantly faster than base R, especially for large datasets. It can be 20-100 times faster for certain operations.
  - dplyr's syntax allows for function chaining, making code cleaner and easier to read and write.
  - dplyr has a set of functions focused on common data manipulation tasks, making it simpler to use
- Cons
  - Some operations, particularly those involving row manipulations, can be simpler in base R
  - New users may find it challenging to learn dplyr's syntax and approach, especially if they're already familiar with base R
  - Base R doesn't require additional package installations

Core dplyr functions

function	description
select()	keep or remove columns (variables)
filter()	keep certain rows
distinct()	deduplicate rows
rename()	rename columns
mutate()	create and transform columns
arrange()	sort rows
recode()	recode levels of a factor
pull()	extract values from a column

Load packages and data

- Install and load packages
  - `> library(tidyverse)`
  - `> library(skimr)`

- Load and inspect data
  - `> library(NHANES)`
  - `> data(NHANES)`

- Task: Try out the function skim. What does it do?
  - `> skim(NHANES)`

Select variables and rows

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

```
> select(NHANES, Age) %>% pull() %>%  mean()
> mean(pull(select(NHANES, Age)))

> NHANES_subset <- select(NHANES, ID, SurveyYr, Gender, Age)
> NHANES_subset <- NHANES %>% select(ID, SurveyYr, Gender, Age) # alternative
> NHANES_subset <- filter(NHANES_subset, row_number() %in% 1:200)
> NHANES_subset <- NHANES %>% select(ID, SurveyYr, Gender, Age) %>%
  filter(row_number() %in% 1:200) # combined
```

Select – helper functions

function	description
everything()	all other columns not mentioned
last_col()	the last column
contains()	columns containing a character string example: select(contains("time"))
starts_with()	matches to a specified prefix example: select(starts_with("date_"))
ends_with()	matches to a specified suffix example: select(ends_with("_post"))
num_range()	a numerical range like x01, x02, x03
any_of()	matches IF column exists but returns no error if it is not found

Select, order, and remove

- Re-order variables
  ```
  > NHANES_subset <- NHANES %>% select(SurveyYr, ID, Age, everything())
  ```
- Select variables
  ```
  > NHANES_subset <- NHANES %>% select(ID, SurveyYr, Gender, Age,
    starts_with(c("BMI", "BP")), contains("Income"), last_col())
  ```
- Remove variables
  ```
  > NHANES_subset <- NHANES_subset %>% select(!"PregnantNow")
  > NHANES_subset <- NHANES_subset %>% select(-c("BPSys3", "BPDia3"))
  ```

Deduplication

- Identify and remove duplicates

```
> NHANES_unique <- NHANES %>% distinct()
> NHANES_unique <- NHANES %>% distinct(ID, SurveyYr)
```

- Why are the resulting unique sets not of the same size?

Rename variables

- Rename the variable – rename(NEW = OLD)

```
> NHANES_new <- NHANES %>%
      rename(Year = SurveyYr,
      Sex = Gender)
```

Generate or modify variables

- Calculate BMI

```
> NHANES_new <- NHANES %>%
    mutate(BMI_new = Weight / ((Height/100)^2)) %>%
    select(ID, Weight, Height, BMI, BMI_new)
```

- Calculate high income

```
> NHANES_new <- NHANES %>%
    mutate(BMI_new = Weight / ((Height/100)^2)) %>%
    mutate(HighIncome = if_else(HHIncomeMid > 75000, 1, 0)) %>%
    select(ID, Age, Weight, Height, BMI, BMI_new, HHIncomeMid, HighIncome)
```

- Modify high income

```
> NHANES_new <- NHANES_new %>%
    mutate(HighIncome = if_else(HHIncomeMid > 50000, 1, 0))
```

Convert and re-code

- Convert format

```
> NHANES_new <- NHANES_new %>%
     mutate (ID = as.character(ID),
      HighIncome = as.factor(HighIncome))
```

- Recode variables – recode(…, OLD = NEW)

```
> NHANES_new <- NHANES_new %>%
     mutate(HighIncome = recode(HighIncome, "0" = "No", "1" = "Yes"))
```

- Task: Calculate an indicator variable for obesity (BMI>30)
  – How many survey participants are obese?

Categorize – case_when()

- Generate age in 20-year bands

```
> NHANES_new <- NHANES_new %>%
    mutate(AgeBand = case_when(
      Age < 20 ~ "0-19",
      Age < 40 ~ "20-39",
      Age < 60 ~ "40-59",
      Age < 80 ~ "60-79",
      Age >= 80 ~ "80+"
    ))
```

Filter rows

- Filter subgroups

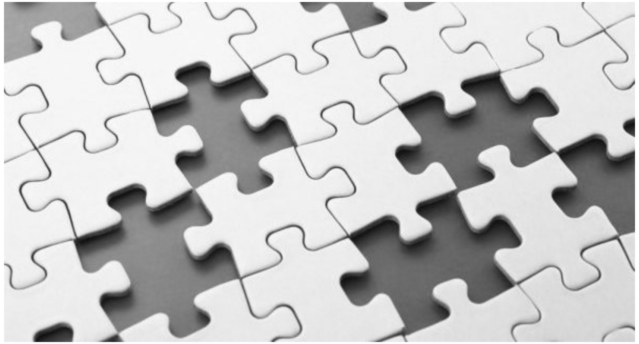
```
> NHANES_subset <- NHANES %>%
      filter(Gender == "female")
```

- Filter out missing values

```
> NHANES_subset <- NHANES %>%
      filter(!is.na(Education))
> NHANES_subset <- NHANES %>%
      drop_na(Education, HHIncome)
```

Missing values

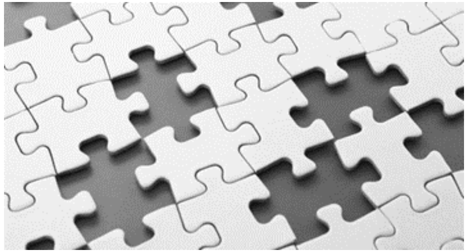
- What are pitfalls when dealing with missing values?



Missing values

- What are pitfalls when dealing with missing values?
  - Some functions don’t work if missing values are present
  - Some functions delete observations if missing values are present. If many variables are used, this can lead to big data loss.
  - Deleting missing data can introduce bias.
  - Many statistical and machine learning models cannot handle missing values.
  - Caveat: Sometimes missing values are coded as 0 or 999 and must be set to NA (or a missing category).
  - Check NAs carefully when filtering data (some functions include NAs, some exclude them such as filter()).
  - Check NAs carefully when merging data

Missing values in R



- Missing values in R are typically represented by NA (Not Available)
- Use is.na() function to identify NA values
- sum(is.na()) can count total NA values in a dataset
- Many R functions have an na.rm parameter. Setting na.rm = TRUE ignores NA values during calculations
  - Example: mean(x, na.rm = TRUE) calculates the mean excluding NA values

- How many missing values are there in the variable BMI?

Sort data

- Ascending

```
> NHANES_new <- NHANES_new %>%
      arrange(Weight)
```

- Descending

```
> NHANES_new <- NHANES_new %>%
      arrange(desc(Weight))
```

- Nested

```
> NHANES_new <- NHANES_new %>%
      arrange(desc(AgeBand), Weight)
```

Summary tables

- The {gtsummary} package provides an elegant and flexible way to create publication-ready summary tables.
- Perfect for presenting descriptive statistics, comparing group demographics (e.g., creating a Table 1 for medical journals), and more.
- Automatically detects continuous, categorical, and dichotomous variables, calculates appropriate descriptive statistics, and also includes amount of missingness in each variable.
- Customizes tables using a growing list of formatting/styling functions. Bold labels, italicize levels, and add p-value.

Summary tables

- Summary table

```
> library(gtsummary)
> table1 <- NHANES_new %>%
      tbl_summary(include = c(Age, AgeBand, BMI, HighIncome))
```

- by HighIncome

```
> table1 <- NHANES_new %>%
      tbl_summary(include = c(Age, AgeBand, BMI), by = HighIncome)
      %>% add_p()
```

- save as HTML

```
> library(gt)
> gtsave(as_gt(table1), filename = "Table1.html")
```

Exercise

Characteristic	Overall N = 3,722[1]	female N = 1,882[1]	male N = 1,840[1]	p-value[2]
Age	40 (30, 52)	40 (30, 51)	41 (30, 52)	0.7
Race				0.015
Black	581 (16%)	317 (17%)	264 (14%)	
Hispanic	284 (7.6%)	151 (8.0%)	133 (7.2%)	
Mexican	448 (12%)	199 (11%)	249 (14%)	
White	2,066 (56%)	1,033 (55%)	1,033 (56%)	
Other	343 (9.2%)	182 (9.7%)	161 (8.8%)	
CivilStatus				0.10
Single	1,435 (39%)	750 (40%)	685 (37%)	
Partner	2,287 (61%)	1,132 (60%)	1,155 (63%)	
Education				<0.001
8th Grade	245 (6.6%)	113 (6.0%)	132 (7.2%)	
9 - 11th Grade	481 (13%)	215 (11%)	266 (14%)	
High School	779 (21%)	371 (20%)	408 (22%)	
Some College	1,171 (31%)	614 (33%)	557 (30%)	
College Grad	1,046 (28%)	569 (30%)	477 (26%)	