C91AR | ADVANCED STATISTICS USING R

Computer Lab 4: Tidying realistic data

Dr Pete McKenna

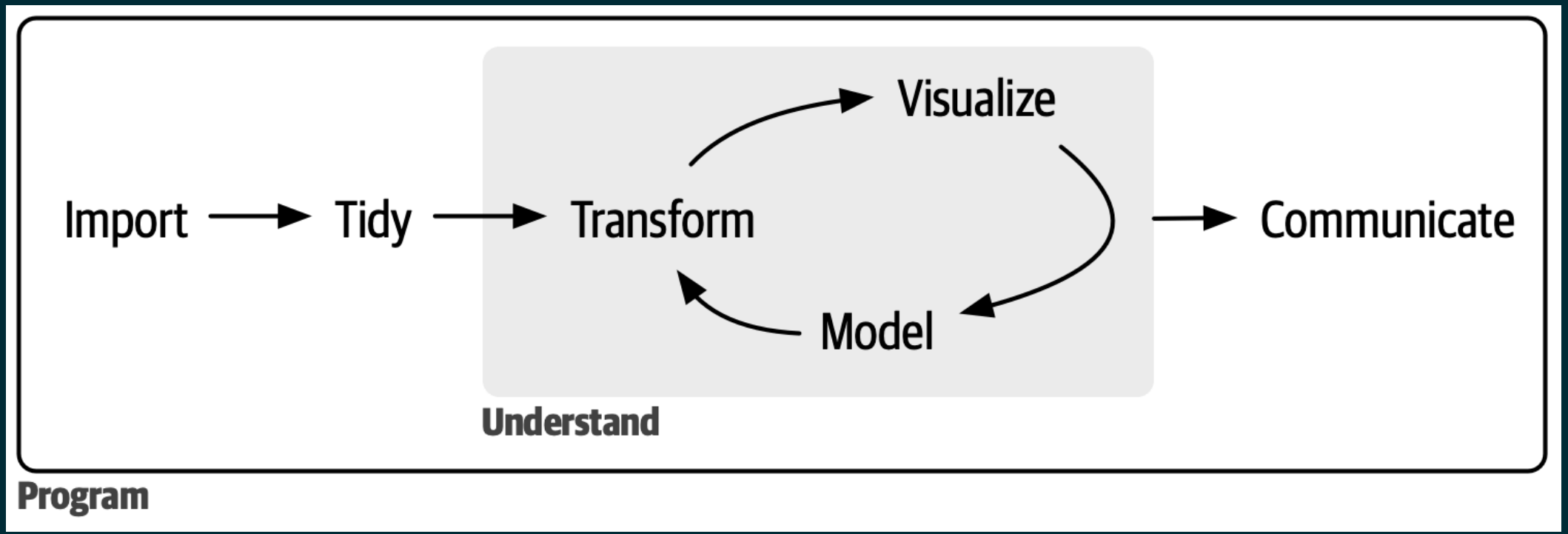
2025-02-05

HERIOT WATT UNIVERSITY
SoSS Doctoral Centre

SETUP FOR THIS MODULE

- Continue working on R Markdown called “Lecture4_tiding_real_data” (or create this file if you weren’t here on Monday)
- Load the following packages

```
# load packages
pacman::p_load(tidyverse,
               snakecase,
               psych,
               summarytools)
```



HERIOT WATT UNIVERSITY
SoSS Doctoral Centre

SAVE THE RAW DATA PROVIDED

- Save the data from Canvas Module **Course datasets** called “robot-expression-data_raw.csv” into the “data_raw” folder

READ IN AND EXPLORE THE ROBOT EXPRESSION DATA

```
1  # Read in raw data
2  df <-
3    read_csv("data_raw/robot-expression-data_raw.csv")
```

EXPRESSION DATASET METADATA

- `ID_` = participant id
- `sex` = participant sex
- `eth` = participant ethnicity
- `eng` = “Is English your native language?”
- `Exp.date` = date of experiment
- `interact.before` = “Have you ever interacted wtih a robot before?
- `Alyx.is` = participants guess of robot’s gender
- `item` = plastic food item offered to robot
- `Expression` = a numeric value for each of the four robot facial expressions
- `resp` = name of the box where participants placed the food item after robot expression
- `Time` = robots onboard clock timer
- `resp_acc` = accuracy of response according to theoretically deigned expressio

HERIOT WATT UNIVERSITY
SoSS Doctoral Centre

DATA TYPES FOR THE `col_types` ARGUMENT

- “f” = factor (i.e., categorical variable)
- “l” = logical
- “i” = integer (no decimals)
- “d” = double numeric (has decimals)
- “c” = character
- “D” = date
- “_” = skip
- “?” = guess

- For more information see the `read_csv` and `col_types` documentation in the help section
- You can pull this up by running `?read_csv` into the console
  - I’d recommend consulting the descriptor info at the top then skipping to the examples at the bottom of the help sheet

SETTING COL TYPES WITHIN THE `read_csv` FUNCTION

```
# Read in data and supply data type argument
df <-
  read_csv("data_raw/robot-expression-data_raw.csv",
           col_types = "ffff?ffffftf") # this is good for Base R

df <-
  read_csv("data_raw/robot-expression-data_raw.csv",
           col_types = "cccc?ccccctc") # tidyverse prefers character vecto
```

- Now the data from the csv is stored in an object called `df`

DATA EXPLORATION

- Here are a few handy functions to exploring your data
  - `summary` : from base R
  - `glimpse` : from *tidyverse*
  - `headTail` : from *psych*
  - `dfSummary`: from *summarytools*

SUMMARISING DATA FUNCTION

- The `summary` function summarises continuous and categorical vectors
- NOTE: `summary` will not count character vectors, that's why we specified that our categorical vectors were factor type (e.g., "f") at `read_csv`

```
# data summary - overall
df |>
  summary()

# structure of the data
glimpse(df)
```

- Why is `glimpse` especially useful having used the `col_types` with `read_csv`?

`psych::headTail` FUNCTION

- The `headTail` function from the *psyche* package

```
1  # data summary - head
2  headTail(df)
```

`summarytools` PACKAGE

- We've seen how we can get an overview of the data using `dfSummary`
- This function is particularly useful for spotting anomalies in the data

```
1  dfSummary(df)
```

TIDYING VECTOR LABLES (I.E., COLUMN NAMES)

- We can use the `snakecase` package function `to_snake_case` to apply uniform *snake case* formatting to column names
- This addresses the point made earlier about consistent variable names

```
# check application of snake_case to data
to_snake_case(names(df))

# irreversibly apply snake_case to vector labels
names(df) <-
  to_snake_case(names(df)) # lower_case with underlined separators

# examine the data
names(df)
```

CHANGING VECTOR NAMES

- You can also change the names of vectors using the `rename` function
- Note, the `rename` argument asks for the **new vector label followed by the old one**
  - this takes a little getting used to but its simple after enough practice

```
1  # Rename vectors to more descriptive labels
2  df |>
3    rename(perceived_robot_gender = alyx_is,
4           robot_expression       = expression,
5           ppt_acc                = resp_acc) |>
6    print()
```

- Using the code `names(df)` in the console, examine the vector labels. What does the output tell you?

RECODING VARIABLE LEVELS

- You may have noticed from the last session that there there are some erroneous values in the sex vector.
- You can check this directly by using the `$` symbol to index the vector of interest in combination with the function `unique`.

```
# Check all unique entries in the vector `sex`
unique(df$sex)

# Alternatively you can use tidyverse's `distinct`
df |>
  select(sex) |>
  distinct()
```

- What does `select` do here?
- As you can see, something has gone wrong in the labelling of sex: there should only values, “Female” and “Male”

USING `case_match` TO RECODE DATA

- Previously, the `recode` function was used in combination with `mutate` to recode values in a cell
- But more recently, people have moved to using `case_match`, as it is more intuitive and user friendly
- Info on case_match

case_match SYNTAX

```
# Using case_match to recode cell values
df <-
  df |>  # create new and tidy object for illustration
  mutate(sex = case_match(sex,                # mutate to change
                          "male" ~ "Male",    # supply erroneous value and r
                          "ffemale" ~ "Female",
                          "femalew" ~ "Female",
                          .default = sex))    # preserve all other entries i

# Check the values of a vector using unique
unique(df$sex) # again, we use the $ symbol to isolate a specific vector
```

HERIOT WATT UNIVERSITY
SoSS Doctoral Centre

FOR LARGER DATASETS

- If you wanted to match values by a pattern, you can use `stringr` commands to recode variable levels
- In this case, we use the function `case_when` and `str_detect` to search the `sex` vector for values starting with either capital or lower case “m” and replace it with “Male”, and the same for principle “f”, to be recoded as “Female”

```
# Using case_when and str_detect to modify rows by string patterns
df |>
  mutate(sex =
           case_when(str_detect(sex,
                                "^[Mm]") ~ "Male",
                     str_detect(sex,
                                "^[Ff]") ~ "Female",
                     TRUE ~ sex))
```

HERIOT WATT UNIVERSITY
SoSS Doctoral Centre

DEALING WITH THE DATE COLUMN

- Dates are notoriously difficult to deal with in R and our date vector could not be read properly
- Thankfully there is a really useful package that simplifies everything; the aptly named `lubridate`
- It is already loaded as part of the Tidyverse
- Take a look at lubridate here

`lubridate` TO SORT OUT DATE COLUMN

- For more information see the Tidyverse lubridate page

```
df <-
  df |>
  mutate(exp_date = dmy(exp_date)) # mutate is a function that allows you

# Examine data
glimpse(df)
```

- The date column should now be properly machine ready formatted (i.e., YYYY-MM-DD), based on the input formatting (i.e., dd/mm/yyyy)
- The machine readable format YYY-MM-DD is preferable for almost all programming languages.
- In this format, you can ask R to make nuanced calculations between dates.

HERIOT WATT UNIVERSITY
SoSS Doctoral Centre

RECAP ON THE ASSIGN <- OPERATOR CONSIDERATIONS

- Irreversibly overwriting dataframe/tibble

```
df <-
  df |>
  select(id, eth)

# df only contains data about ppt id and eth
```

Create new dataframes/tibbles

```
df1 <-
  df |>
  select(id, eth)

# df1 only contains data about ppt id and eth
# df contains all 12 vectors
```

HERIOT WATT UNIVERSITY
SoSS Doctoral Centre

INTRODUTION DATA WRANGLING

- You can create *tidyverse* scripts that summarise data similarly to `psych`
- `psych` is better for numeric data, which we'll be using next week.

```
df |>
  group_by(sex,
           resp_acc) |>
  summarise(n = n()) |>
  mutate(freq = n/sum(n))|>
  mutate(freq = round(freq*100, digits = 2))
```

- Having had a look at the output, which is the above code doing? Think about each step of the process, punctuated by `|>`

HERIOT WATT UNIVERSITY
SoSS Doctoral Centre

EXERCISE

- What code would you write to calculate the proportion of different ethnicities and their prior experience with robots?
- How would we create a tibble called `df_non_native` that only included non-native English speakers in the data?

SAVE YOUR TIDY DATA

- Once you are satisfied with the amendments made to your data you can save it as a new tidy `.csv` file
- Again, use comments and bookmarks to signpost this part of your script

```
# save your new data object into `data_tidy`
write_csv(df, # name of the object
          "data_tidy/robot-expression-data_tidy.csv") # desired directory
```

- Note: you can provide any name you like for the file in here, the important part is to pass `read_csv` the tibble of interest, and to include `data_tidy` in the stated directory

HERIOT WATT UNIVERSITY
SoSS Doctoral Centre

CLEANUP

```
 1  # Clear data
 2  rm(list = ls())   # Removes all objects from environment
 3
 4  # Clear packages
 5  p_unload(all)   # Remove all contributed packages
 6
 7  # Clear plots
 8  graphics.off()   # Clears plots, closes all graphics devices
 9
10  # Clear console
11  cat("\014")   # Mimics ctrl+L
```

REFERENCES

HERIOT WATT UNIVERSITY

SoSS Doctoral Centre


HERIOT
WATT
UNIVERSITY
SoSS Doctoral
Centre