# Essential Research Toolkit for the Humanities

## Week 4: Looking at data

Anna Pryslopska

April 29, 2024

Psycholinguistics and Cognitive Modeling Lab

# Homework

1. According to R, what is the type of the following:

`typeof(1L) or is.numeric(1)`

| | |
|---|---|
| `"Anna"` | character |
| `-10` | integer |
| `FALSE` | logical |
| `3.14` | double |
| `as.logical(1)` | logical |

2. According to R, is the following true:

| | |
|---|---|
| ✔ `7+0i == 7` | numeric |
| ✔ `9 == 9.0` | numeric |
| ✖ `"zero" == 0L` | character != numeric |
| ✔ `"cat" == "cat"` | exactly the same |
| ✔ `TRUE == 1` | TRUE evaluates to 1 |

3. What is the output of the following operations and ❗ why ❗?

| | | |
|---|---|---|
| ✖ | `10 < 1` | 10 > 1 |
| ✔ | `5 != 4` | aren't the same |
| **5** | `5 - FALSE` | **FALSE** is 0 |
| ✔ | `1.0 == 1` | same number |
| ✔ | `4 * 9.1` | multiplication |
| ⊘ | `"a" + 1` | character != numeric |
| **NaN** | `0/0` | can't divide 0 by 0 |
| ⊘ | `b * 2` | no variable **b** |
| **-Inf** | `(1-2)/0` | negative / 0 = tends towards negative infinity |
| ⊘ | `10 <- 20` | can't assign to number |
| ✖ | `NA == NA` | comparison not meaningful |
| ✖ | `-Inf == NA` | comparison not meaningful |

## 4. Read and inspect the `noisy.csv` data. What are the meaningful columns? What should be kept and what can be discarded?

❝ It is difficult to decide which columns are meaningful and which are not without the documentation provided. ❞

I didn't expect you to know this, but to take an educated guess.

❝ Based just by the names of the columns,... ❞

Look at the data, names can be misleading or meaningless.

❝ [I] get `"Error in read_csv("noisy.csv") : could not find function "read_csv"` ❞

No function → load `tidverse` package

Ask in the forum for help.

4. Read and inspect the **`noisy.csv`** data. What are the meaningful columns? What should be kept and what can be discarded?

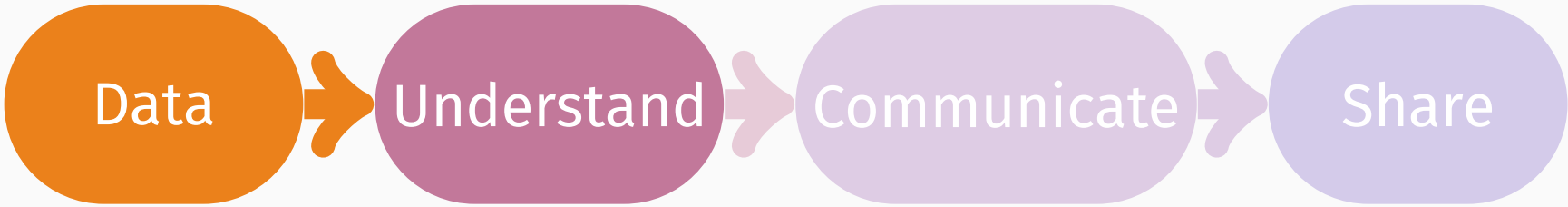
| | |
|---|---|
| `id` or `MD5.hash.of.part…` | unique participant identifier |
| `Label` | experiment section (instructions, experiment, etc.) |
| `PennElementType` | action type (reading, answering, etc.) |
| `Parameter` | single trial part (what is being shown) |
| `Value` | parameter detail (sentence part or judgment) |
| `ITEM` | sentence numer |
| `CONDITION` | sentence variant |
| `Reading.time` | time spent reading |
| `Sentence..or.sentence.MD5.` | whole sentence |
| `(EventTime, Results.reception.time)` | timestamp |
| `(Native, german)` | demographic info |

Questions?

# Table of contents

# Where are we this week?

# Recap




R & RStudio, packages, data types, formats, encoding

import from workspace, assign values, operations, clean, filter, arrange, select, merge, group, summarize, export, visualize

document, create clean and beautiful reports

connect, collaborate, backup

# Cleaning and transforming data


CLEAN ALL THE
THINGS!

# Renaming

```
colnames(moses)

rename(moses,
       ID = MD5.hash.of.participant.s.IP.address,
       ANSWER = Value)
```

# Selecting

Tidyverse

```
select(moses, ID, ITEM, CONDITION, ANSWER)
select(moses, c(ID, ITEM, CONDITION, ANSWER))
select(moses, c(ID, ITEM:ANSWER))
```

base R

```
moses$ID
moses[ , "ID"]
moses[ , c("ID", "ITEM", "CONDITION", "ANSWER")]
moses[ , c(1,4:6)]
subset(moses, subset = ITEM == 1, select = c(ID))
```


- ✔ `10 < 1` — return/show
- ✔ `print(10 < 1)` — print
- ✔ `c(10 < 1)` — concatenate
- ✔ `cat(10 < 1)` — concatenate & print

# Task 1: Rename and drop columns

Look at the dfferent columns in the `moses` data frame. Create a new data frame with the following changes:

1. Change the column name `MD5.hash.of.participant.s.IP.address` to `ID` and `Value` to `ANSWER`
2. Create a new data frame from the previous one with the columns: `ITEM, CONDITION, ANSWER, ID, Label, Parameter`

```
rename(WHERE, WHAT)
```

```
select(WHERE, WHAT)
```

# Missing data

```
na.omit(moses)                    everywhere
na.omit(moses$Item)               only in the items
na.omit(moses[ , "Item"])         only in the items
na.omit(moses[ , 4])              only in the items
```

```
is.na(WHERE)
is.na(select(moses, Item))
```

# Task 2: Remove missing values

Create a new data frame from the previous one with no NAs anywhere.

`na.omit(WHERE)`

# Coding basics: R as a calculator

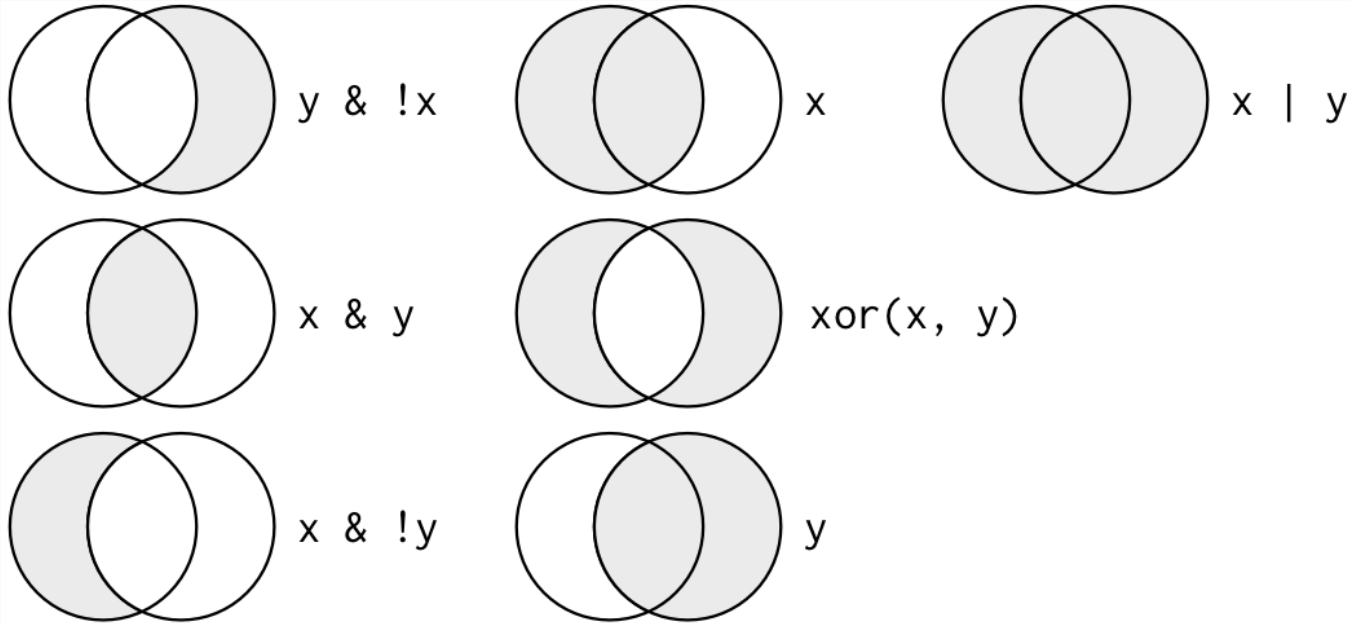
| | |
|---|---|
| addition | `+` |
| subtraction | `-` |
| division | `/` |
| multiplication | `*` |
| power | `^` |
| equals | `==` |
| not equals | `!=` |
| greater than | `>` |
| greater than or equal | `>=` |
| less than | `<` |
| less than or equal | `<=` |
| range | `NUMBER1:NUMBER2` |
| identify element | `VALUE %in% OBJECT` |

# Coding basics: Logic

negation !

logical *and* &

logical *or* |




Wickham et al. (2023)

# Filter (out)

```
filter(moses, CONDITION == 1)
```
condition 1

```
filter(moses, CONDITION %in% 1)
```
condition 1

```
filter(moses, CONDITION >= 1 & Condition < 2)
```
condition 1

```
filter(moses, CONDITION == 1 | CONDITION == 2)
```
conditions 1–2

```
filter(moses, CONDITION %in% 1:2)
```
conditions 1–2

```
filter(moses, CONDITION < 100)
```
conditions 1–2

```
filter(moses, CONDITION %in% c(1, 2))
```
conditions 1–2

```
filter(moses, CONDITION == 1:2)
```
condition 1, if data is duplicated, then 1–2

# Task 3: Remove unnecessary rows

Create a new data frame from the previous one with rows that fulfill the following conditions. Keep the other columns.

1. `Parameter` is ”Final”
2. `Label` is NOT ”instructions”
3. `CONDITION` is only 1 or 2

```
filter(WHERE, TRUE CONDITION, TRUE CONDITION)
```

# (Re)arrange

| | |
|---|---|
| `arrange(moses, ITEM)` | item |
| `arrange(moses, ITEM, CONDITION)` | item, then condition |
| `arrange(moses, desc(ID))` | ID, descending |
| `arrange(moses, desc(is.na(ANSWER)))` | |

# Task 4: Sort the values

Create a new data frame from the previous one in which the values are sorted by `ITEM` and `CONDITION` in ascending order, and `ANSWER` in descending order.

```
arrange(WHERE, HOW)
```

# Create and mutate

Tidyverse

```
mutate(moses, CLASS = TRUE)
```
make new column

```
mutate(moses, NUMBER = 1:598)
```

```
mutate(moses, ITEMS = ITEM + 1)
```
calculate column

```
mutate(moses, ITEM1 = ITEM == 1)
```
evaluate column

```
mutate(moses, CONDITION = as.character(CONDITION))
```
overwrite column

```
mutate(moses, ITEM1 = NULL)
```
remove column

# Create and mutate

base R

```
moses$CLASS <- TRUE
moses$NUMBER <- 1:598
moses$ITEMS <- moses$ITEM + 1
moses$ITEM1 <- moses$ITEM == 1
moses$CONDITION <- as.character(moses$CONDITION)
moses$ITEM1 <- NULL
```

❗ Assignment saves, so be careful! This code deletes `ITEM1` and permanently changes `CONDITION`.

# Task 5: Re-code item number

Create a new data frame from the previous one, but with the `ITEM` column being numeric. Look at the first 20 rows of this new data frame.

```
mutate(WHAT, NEW = CHANGE HOW)
```

# Re-code inconsistent information

```
unique(VALUES)                                  show all unique values
unique(moses$ANSWER)                                        plain list
unique(select(moses, ANSWER))          only as many as fit on screen
print(unique(select(moses, ANSWER)), n=Inf)                 show all
```

# Task 6: Look at possible answers

Look at all answers give the the question with the item number 2.
Save all unique answers to a new variable.

`filter`, `select`, `unique`, assign

# Pipes

# Pipes

# Pipes

Powerful tool for clearly expressing a sequence of multiple operations. Passes the output as the new input.

Created using `|>` (base) or `%>%` (magrittr). They can be read as “and then”.

The pipe translates `x |> f(y)` into `f(x, y)`.

```
moses_clean1 <- na.omit(moses)
moses_clean2 <- select(moses_clean1, CONDITION)
moses_clean3 <- filter(moses_clean2, CONDITION %in% 1:2)
moses_clean4 <- arrange(moses_clean3, CONDITION)

moses_clean4 <- moses |>
                na.omit() |>
                select(CONDITION) |>
                filter(CONDITION %in% 1:2) |>
                arrange(CONDITION)
```


Why? Simplify code, remove clutter and potential for error, reduce effort, stay reproducible.

Why not? No intermediate steps (need to run the whole code), writing functions is more complex.

When not? Very long pipes (>10 lines), multiple inputs or outputs, creating plots.

# Wrap-up

# Summary

- ✔ renaming
- ✔ selecting
- ✔ dealing with missing data
- ✔ coding basics
- ✔ logic
- ✔ filtering
- ✔ arranging
- ✔ pipes
- ⏩ Mutating, grouping, summarizing, getting help, tidy code

# Homework assignment due May 3rd 15:30

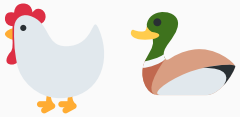
Submit 1 R script.

- Complete logic exercise (next slide)
- Complete assignment 3 ($\rightarrow$ ILIAS)

# Logic exercise

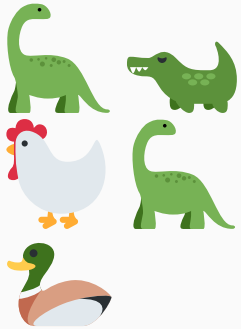
Your world has four individuals:

Two are of the type `bird`

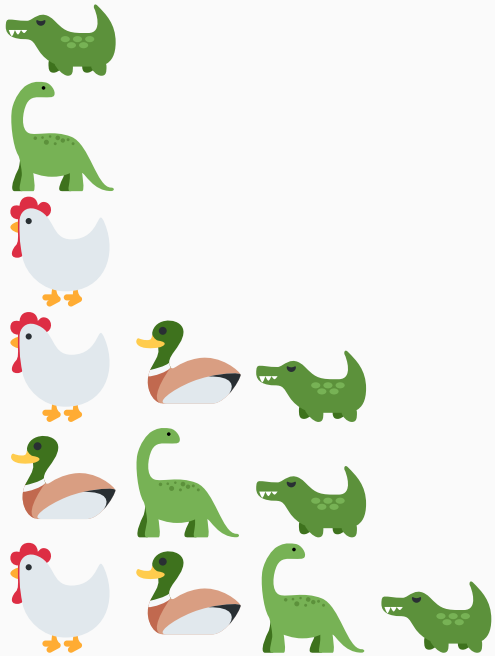


Two are of the type `can swim`

Using only basic logical expressions (negation `!`, and `&`, or `|`) and the two groups, describe the groups on the right, as in the first example. Tip: a Venn diagram as on slide 15 might help.



`!bird`



∅ (i.e. exclude all)

# Clean up on aisle “Answer”

```
cant_answer <- c("Can't Answer", "Can't answer",
"Can't answer the question", "Can't answrer",
"Can't be answered", "Can´t answer", "i can't
answer", "can't andwer" , "can't answer" , "can't
answer (Nobel is given by Norway)", "can't asnwer",
"can't know", "can`t answer", "can`t asnwer" ,
"cant answer", "can´t answ", "can´t answer", "no
answer")
```

ℹ use `arrange()`, `filter()`, `select()`, and `unique()`

# References


Wickham, Hadley, Mine Çetinkaya-Rundel, and Garrett Grolemund (2023). *R for data science: import, tidy, transform, visualize, and model data*. 2nd ed. O'Reilly Media, Inc. URL: https://r4ds.hadley.nz/.