Data communication with tables

Data dictionaries and summary tables

Daniela Palleschi

2023-04-13

Table of contents

Learning objectives

- create a data dictionary
- produce formatted tables with the `knitr` package
- produce summary tables

Load packages and data

```
# load tidyverse
pacman::p_load(tidyverse, here)

# load data
df_lifetime <- readr::read_csv(here::here("data/tidy_data_lifetime_pilot.csv"),
                               # for special characters
                               locale = readr::locale(encoding = "latin1")
                               ) |>
  mutate_if(is.character,as.factor) |> # all character variables as factor
  filter(type == "critical", # only critical trials
         px != "px3") # this participant had lots of 0's for some reason
```

Data dictionary

- we haven't really discussed what exactly our data *are*, though
- data dictionaries (a.k.a. code books)
  - describe each variable in a dataset
  - ideally also provide information regarding possible values

Variable names

- we can list the names of all variables in a dataset using `names()`

```
names(df_lifetime)
```

```
 [1] "px"            "trial"         "region"        "region_n"
 [5] "region_text"   "eye"           "ff"            "fp"
 [9] "rpd"           "tt"            "fix_count"     "reg_in"
```

```
[13] "reg_in_count"  "reg_out"       "reg_out_count" "rt"
[17] "bio"           "critical"      "gender"        "item_id"
[21] "list"          "match"         "condition"     "name"
[25] "lifetime"      "tense"         "type"          "yes_press"
[29] "KeyPress"      "accept"        "accuracy"      "px_accuracy"
```

- but we need to be able to put these names into a single column
  - where each row contains one variable name
  - and other columns contain information like description and data class

Names to rows

```
# From day 2 of Lisa DeBruine's [Coding Club: Creating an R Package](https://psyteachr.githul

# create as many empty strings as we name variable names
coldesc <- rep("", ncol(df_lifetime))
# add variable names to these empty strings
names(coldesc) <- names(df_lifetime)
```

```
# print as code needed to create an object
dput(coldesc)
```

```
c(px = "", trial = "", region = "", region_n = "", region_text = "",
eye = "", ff = "", fp = "", rpd = "", tt = "", fix_count = "",
reg_in = "", reg_in_count = "", reg_out = "", reg_out_count = "",
rt = "", bio = "", critical = "", gender = "", item_id = "",
list = "", match = "", condition = "", name = "", lifetime = "",
tense = "", type = "", yes_press = "", KeyPress = "", accept = "",
accuracy = "", px_accuracy = "")
```

- copy the output of `dput(coldesc)` and assign it to an object
  - tip: you can reformat the code by highlighting it and using `Cmd/Ctrl+Shift+A`
    - or in the menu bar: Code > Reformat Code
- replace `c()` with `tibble()` to create a dataframe
  - and fill in the quotations with description of the data

```
dict_lifetime <- tibble(
  px = "participant ID (factor)",
  trial = "trial number (ordered factor)",
  region = "sentence region (order factor)",
  region_n = "numerical representation of sentence region (ordered factor)",
  region_text = "text presented in the region (string)",
  eye = "which eye was tracking: right or left (binomial)",
  ff = "first-fixation times in milliseconds (continuous, values can be 0<)",
  fp = "first-pass reading times in milliseconds (numeric, values can be 0<)",
  rpd = "regression-path duration in milliseconds (numeric, values can be 0<)",
  tt = "total reading time in milliseconds (numeric, values can be 0<)",
  fix_count = "number of total fixations in the region (count)",
  reg_in = "whether of a regression was made into the regions (binomial: 0 = no, 1 = yes)",
  reg_in_count = "number of fixations into the region (count)",
  reg_out = "whether of a regression was made out of the regions (binomial: 0 = no, 1 = yes)"
  reg_out_count = "number of fixations out of the region (count)",
  rt = "reaction time from critical sentence presentation to button press (continuous, values
  bio = "lifetime biography context sentence (string)",
  critical = "critical sentence (string)",
  gender = "gender of stimulus subject (binomial: male, female)",
  item_id = "item identification number (critical items: 1-80)",
  list = "experimental list version: base list version (1-4) and whether the yes-button was
  match = "whether the referent-lifetime was congruent with tense",
  condition = "condition: lifetime (dead, alive) + tense (PP, SF) (factor)",
  name = "name of stimilis subject (string)",
  lifetime = "lifetime status of stimulus subject at time of experiment (binomial: dead, ali
  tense = "tense used in critical sentence (binomail: PP = present perfect, SF = simple futu
  type = "sentence type (factor with one level: critical)",
  yes_press = "corresponding coding for the yes-button on Cedrus response box (4 = left butt
  KeyPress = "key that was pressed (4 = left button, 5 = right button)",
  accept = "whether the item was accepted, i.e., whether KeyPress equalled yes_press",
  accuracy = "whether the acceptance was accurate (reject for a mismatch, accept for a match
  px_accuracy = "participant's overall accuracy score"
)
```

- but `dict_lifetime` doesn't have the shape we want
  - each variable name is a column name
  - and its description is in the first row

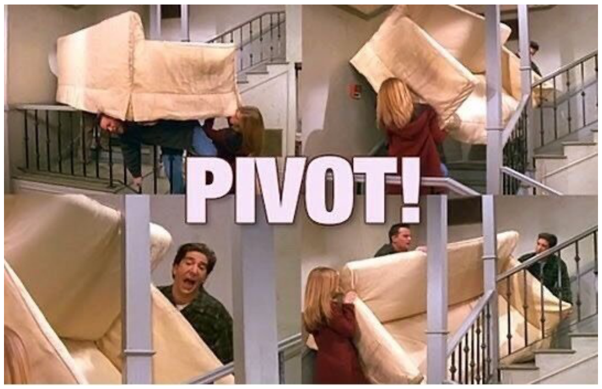
```r
dict_lifetime
```

```
# A tibble: 1 x 32
  px             trial region region_n region_text eye   ff    fp    rpd   tt   
  <chr>          <chr> <chr>  <chr>    <chr>       <chr> <chr> <chr> <chr> <chr>
1 participant I~ tria~ sente~ numeric~ text prese~ whic~ firs~ firs~ regr~ tota~
# i 22 more variables: fix_count <chr>, reg_in <chr>, reg_in_count <chr>,
#   reg_out <chr>, reg_out_count <chr>, rt <chr>, bio <chr>, critical <chr>,
#   gender <chr>, item_id <chr>, list <chr>, match <chr>, condition <chr>,
#   name <chr>, lifetime <chr>, tense <chr>, type <chr>, yes_press <chr>,
#   KeyPress <chr>, accept <chr>, accuracy <chr>, px_accuracy <chr>
```

- we want to transpose the data
  - i.e., rotates the data so that the column names are in a row, with the descriptions in another row

`pivot_longer()`

- takes wide data and makes it longer
  - converts headers of columns into values of a new column
  - combines the values of those columns into a new condensed column
- takes a few arguments:
  - `cols`: which columns do we want to combine into a single column?
  - `names_to`: what should we call the new column containing the previous column names?
  - `values_to`: what should we call the new column containing the values from the previous columns?



Pivot our data dictionary

- this looks much better!

```
dict_lifetime <-
  dict_lifetime |>
  pivot_longer(
    cols = everything(),
    names_to = "variable",
    values_to = "description"
  )
```

Save data dictionary

- now we can save our data dictionary just like we would any csv

```
write_csv(dict_lifetime, here("data", "tidy_data_lifetime_pilot_dictionary.csv"))
```

Formatted tables

- when we render our document, `dict_lifetime` won't look very pretty
- there are several packages that produce nicely formatted tables
  - `knitr`

```r
dict_lifetime |>
  knitr::kable()
```

variable	description
px	participant ID (factor)
trial	trial number (ordered factor)
region	sentence region (order factor)
region_n	numerical representation of sentence region (ordered factor)
region_text	text presented in the region (string)
eye	which eye was tracking: right or left (binomial)
ff	first-fixation times in milliseconds (continuous, values can be 0<)
fp	first-pass reading times in milliseconds (numeric, values can be 0<)
rpd	regression-path duration in milliseconds (numeric, values can be 0<)
tt	total reading time in milliseconds (numeric, values can be 0<)
fix_count	number of total fixations in the region (count)
reg_in	whether of a regression was made into the regions (binomial: 0 = no, 1 = yes)
reg_in_count	number of fixations into the region (count)
reg_out	whether of a regression was made out of the regions (binomial: 0 = no, 1 = yes)
reg_out_count	number of fixations out of the region (count)
rt	reaction time from critical sentence presentation to button press (continuous, values can be 0<)
bio	lifetime biography context sentence (string)
critical	critical sentence (string)
gender	gender of stimulus subject (binomial: male, female)
item_id	item identification number (critical items: 1-80)
list	experimental list version: base list version (1-4) and whether the yes-button was coded as 4 or 5 (factor: 14, 15, 24, 25, 34, 35, 44, 45)
match	whether the referent-lifetime was congruent with tense
condition	condition: lifetime (dead, alive) + tense (PP, SF) (factor)
name	name of stimulis subject (string)
lifetime	lifetime status of stimulus subject at time of experiment (binomial: dead, alive)
tense	tense used in critical sentence (binomail: PP = present perfect, SF = simple future)
type	sentence type (factor with one level: critical)
yes_press	corresponding coding for the yes-button on Cedrus response box (4 = left button, 5 = right button)
KeyPress	key that was pressed (4 = left button, 5 = right button)
accept	whether the item was accepted, i.e., whether KeyPress equalled yes_press
accuracy	whether the acceptance was accurate (reject for a mismatch, accept for a match)
px_accuracy	participant's overall accuracy score

Tables as LaTeX code

- you can add the argument "latex" to print LaTeX code for a table in the Console
  - you can then cut and paste this code into a LaTeX (or Overleaf) script

```
dict_lifetime |>
  knitr::kable("latex")
```

- but be careful, if you're rendering to HTML the table won't be printed if you use "latex"

Exercise

1. install the knitr package (install.packages("knitr"))
2. print dict_lifetime, but only for the following variables:
   - px, trial, region_text, ff, fp, and condition
3. use kable() from knitr to print the table

variable	description
px	participant ID (factor)
trial	trial number (ordered factor)
region_text	text presented in the region (string)
ff	first-fixation times in milliseconds (continuous, values can be 0<)
fp	first-pass reading times in milliseconds (numeric, values can be 0<)
condition	condition: lifetime (dead, alive) + tense (PP, SF) (factor)

Data summaries

- we can create summary tables of our data

```
# compute summary
summary_ff <- df_lifetime |>
  filter(region=="verb") |>
  group_by(condition,lifetime,tense) %>%
  summarise(N = n(),
            mean.ff = mean(ff, na.rm = T),
            sd = sd(ff, na.rm = T)) %>%
  # compute standard error, confidence intervals, and lower/upper ci bounds
  mutate(se = sd / sqrt(N),
         ci = qt(1 - (0.05 / 2), N - 1) * se,
         lower.ci = mean.ff - qt(1 - (0.05 / 2), N - 1) * se,
         upper.ci = mean.ff + qt(1 - (0.05 / 2), N - 1) * se)
```

- and print the output with the `kable()` function from the `knitr` package
  - for extra customisation you can also use the `kableExtra` package (e.g., with the `kable_styling()` function)

```
# install.packages("knitr") # if not yet installed
knitr::kable(summary_ff, digits=1,
             caption = "Table with summmary statistics for first-fixation duration at the ve
```

Table 3: Table with summmary statistics for first-fixation duration at the verb region

condition	lifetime	tense	N	mean.ff	sd	se	ci	lower.ci	upper.ci
deadPP	dead	PP	140	198.9	57.9	4.9	9.7	189.2	208.6
deadSF	dead	SF	139	194.6	67.9	5.8	11.4	183.2	205.9
livingPP	living	PP	140	194.2	77.3	6.5	12.9	181.3	207.1
livingSF	living	SF	140	186.0	57.6	4.9	9.6	176.4	195.6

Saving summary tables

- we could also save this table using `write_csv()`
  - but it's relatively simple to re-produce, so I wouldn't bother
  - instead, when writing up my results I would load in the data and print the summary directly
- sometimes summary tables are more code-intensive
  - in this case I would save the summary as a csv, and simply load and print it when writing in R markdown or Quarto

Additional packages

There are many other packages for including tables that are publication-ready. Some that I would suggest you look into:

- kableExtra which includes additionally formatting options for `knitr::kable()` tables via the `kable_styling()` function and others
  - tables must first pass through `knitr::kable()`, e.g., `my_table |> knitr::kable() |> kableExtra::kable_styling()`
- flextable
  - very flexible package for creating publication-ready tables of many formats
- `papaja::apa_table()`: the `papaja` package aids in creating APA-formatted journal articles
  - the `apa_table()` function can take objects containing results from a statisical test/model and output a formatted table
  - we'll discuss this topic more when we get into regression

`kableExtra::kable_styling()`

- we'll first create a little summary table using the `iris` dataset which comes built-in with R

```
sum_iris <- iris |>
  summarise(mean = mean(Sepal.Length),
            sd = sd(Sepal.Length),
            n = n(),
            .by = Species)
```

- and we'll print the summary using `kable_styling()` (Table 4)

```
sum_iris |>
  knitr::kable() |>
  kableExtra::kable_styling()
```

Table 4: Example output of an table using the `kableExtra` package

Species	mean	sd	n
setosa	5.006	0.3524897	50
versicolor	5.936	0.5161711	50
virginica	6.588	0.6358796	50

`flextable`

- print the same summary using `flextable()` (Table 5)

```
sum_iris |>
  flextable::flextable()
```

Table 5: Example output of an table using the `flextable` package

Species	mean	sd	n
setosa	5.006	0.3524897	50
versicolor	5.936	0.5161711	50
virginica	6.588	0.6358796	50

`papaja::apa_table()`

- now run a linear mixed model on the `iris` data

```
lmm_iris <-
  lme4::lmer(Sepal.Width ~ Sepal.Length + Petal.Width +
                (1|Species), data = iris)
```

- and print a model summary table using `apa_table()` (Table 6)

```
lmm_iris |> papaja::apa_print() |>
papaja::apa_table(caption = NULL)
```

Exercise

1. create an object with some summary statistics of the variable `rt`
   - call it `summary_rt`
2. use `kable()` from `knitr` to print a table, it should look something like Table 8
3. try creating the same table with one (or more) of the additional packages we saw above (`kableExtra`, `flextable`, `papaja`)

Table 6: Example output of an LMM using `papaja` package

[tbp]

Table 7

Term	$\hat{\beta}$	95% CI	t
Intercept	0.85	[-0.34, 2.05]	1.39
Sepal Length	0.27	[0.18, 0.36]	5.85
Petal Width	0.51	[0.28, 0.74]	4.37

Table 8: Summary of reaction times (ms) per condition

lifetime	tense	condition	N	mean.rt	sd
dead	PP	deadPP	140	3530.5	2915.8
dead	SF	deadSF	139	1747.0	1153.4
living	PP	livingPP	140	2257.7	1346.3
living	SF	livingSF	140	2578.1	1958.7