Managing Data in R

Data Checking in R

Glenn Williams

University of Sunderland

2020-09-29 (updated: 2020-10-13)

Understanding our Data

To have a good understanding of how you should clean and analyse your data, you have to understand how your data was collected. Ask yourself:

- How was the study performed?
- What do the variables in the data set represent?
- What levels can these variables take?
- Was a consistent coding scheme used throughout?
- How is missing data coded (if at all)?

Reading in Data

Let's assume we performed an experiment. Let's read the data from this experiment into R assuming it is stored as a .csv file:

```r
raw_data <- read_csv(here("data", "factorial_data.csv"))
```

```
## Warning: Missing column names filled in: 'X1' [1]

## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   subj_id = col_character(),
##   list_id = col_double(),
##   item_id = col_double(),
##   A = col_character(),
##   B = col_character(),
##   Y = col_double()
## )
```

Let's take a look at this data so we can understand it...

Some Notes on the Data

Let's take a look at our data. What does it look like? The `head()` function allows you to see the top of your data set. (`tail()` does the opposite.)

```
head(raw_data)
```

```
## # A tibble: 6 x 7
##      X1 subj_id list_id item_id A     B          Y
##   <dbl> <chr>     <dbl>   <dbl> <chr> <chr> <dbl>
## 1     1 one           4       1 A2    B2     234.
## 2     2 one           4       2 A2    B2     295.
## 3     3 one           4       3 A2    B2     285.
## 4     4 one           4       4 A2    B2     258.
## 5     5 one           4       5 A2    B2     268.
## 6     6 one           4       6 A2    B2     282.
```

Let's assume the data is from an experiment.

Next, we'll explain what these mean.

Understanding our Data

Let's assume we gave people a task where they had to read sentences with an upwards or downwards motion (variable A) and then identify targets at the top or bottom of the screen (variable B) as quickly as they could. We captured their response times in the task (variable Y).

It looks like we have the following columns:

- **X1** = Row number for the data.
- **subj_id** = Subject/participant ID for the person who gave the data to us.
- **list_id** = list ID for how items were randomised.
- **item_id** = Item ID.
- **variable A** = Implied sentence location (upwards or downwards); e.g. "the bird flew in the sky."
- **variable B** = Location of target (up or down on the screen).

We should check that all of our data are coded as expected.

Inspecting Data

Our first step to understanding out data should be to check that we have read it into R correctly, and that `read_csv()` has sensibly guessed at our data types.

We can do this using the `glimpse()` function. This gives us a look at our data rotated, so we can see how our data are stored and how they are coded. There's a few problems here...

```
glimpse(raw_data)
```

```
## Rows: 2,400
## Columns: 7
## $ X1       <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
## $ subj_id  <chr> "one", "one", "one", "one", "one", "one", "one", "one", "o
## $ list_id  <dbl> 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4
## $ item_id  <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
## $ A        <chr> "A2", "A2", "A2", "A2", "A2", "A2", "A2", "A2", "A2", "A2"
## $ B        <chr> "B2", "B2", "B2", "B2", "B2", "B2", "B2", "B2", "B2", "B2"
## $ Y        <dbl> 233.5056, 294.6706, 284.6235, 257.9815, 267.9257, 282.1853
```

Identifying Problems

First off, we don't need a count of row numbers (R does this for us), so we can remove the X1 column. Let's do that now.

We'll use the `select()` function from dplyr in the tidyverse.

This takes our data as an argument and any columns we want to select (keep) when named or given a column number. Nicely, we can also use the minus sign to say "keep everything except this".

```
# select out the X1 column from our data, assign this to our new data
cleaned_data <- select(raw_data, -X1)

cleaned_data
```

```
## # A tibble: 2,400 x 6
##   subj_id list_id item_id A     B         Y
##   <chr>     <dbl>   <dbl> <chr> <chr> <dbl>
## 1 one           4       1 A2    B2     234.
## 2 one           4       2 A2    B2     295.
## 3 one           4       3 A2    B2     285.
## 4 one           4       4 A2    B2     258.
## # … with 2,396 more rows
```

Identifying Problems

We can also see that subject IDs are stored as characters. Why is that? Let's see what unique values we have here.

The `unique()` function tells us what unique observations we have in a column.

```
unique(cleaned_data$subj_id)
```

```
##  [1] "one"  "2"    "3"    "4"    "5"    "6"    "7"    "8"    "9"    "10"
## [11] "11"   "12"   "13"   "14"   "15"   "16"   "17"   "18"   "19"   "20"
## [21] "21"   "22"   "23"   "24"   "25"   "26"   "27"   "28"   "29"   "30"
## [31] "31"   "32"   "33"   "34"   "35"   "36"   "37"   "38"   "39"   "40"
## [41] "41"   "42"   "43"   "44"   "45"   "46"   "47"   "48"   "49"   "50"
## [51] "51"   "52"   "53"   "54"   "55"   "56"   "57"   "58"   "59"   "last"
```

OK, we have a pretty terrible naming system here. Surely we should make these all numbers.

Converting Values

Let's make the unique values in our data set for the subject IDs all numbers instead of a mix of numbers and characters.

This looks through the IDs in our data (e.g. "one", "one", ...2, 2,...). Then it checks where in the unique IDs (e.g. "one", 2, 3) it occurs, returning that number.

So, "one" gets a 1, as it's the first unique ID, 2 gets a 2 as it's the second, etc.

Note: this is a complex function; don't worry if this doesn't make sense!

```r
cleaned_data$subj_id <- match(
  cleaned_data$subj_id,
  unique(cleaned_data$subj_id)
)
# check new values
unique(cleaned_data$subj_id)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
## [51] 51 52 53 54 55 56 57 58 59 60
```

Fixing Remaining Problems

Our list IDs and Item IDs are properly coded, but we should check our variables A and B, which indicate the conditions of the study. Which unique values do we have here?

```
unique(cleaned_data$A)
```

```
## [1] "A2"  "A1"  "a1"  "a2"  "Aa1"
```

```
unique(cleaned_data$B)
```

```
## [1] "B2"  "B1"  "b2"  "BB2" "BB1" "b1"
```

It looks like we have a mix of issues here. R is case sensitive, meaning A is different to a. We can make some initial fixes here by making everything the same case. Do this by using the `tolower()` function.

Fixing Remaining Problems

```r
# change values
cleaned_data$A <- tolower(cleaned_data$A)
cleaned_data$B <- tolower(cleaned_data$B)

# check values
unique(cleaned_data$A)
```

```
## [1] "a2"  "a1"  "aa1"
```

```r
unique(cleaned_data$B)
```

```
## [1] "b2"  "b1"  "bb2" "bb1"
```

That looks good, most of the codes are now consistent. But it looks like someone also mistakenly entered e.g. aa1 instead of just a1. The same happens with bb1 and bb2. Let's change these!

Fixing Remaining Problems

We'll need to use some logic here.

We can set up an `ifelse()` statement where we tell R to do something if it finds a match to our request, but that can get repetitive. Instead, one option is to use `case_when()`.

We give this function an argument relating to what **case** we care about (e.g. when A is "aa1") then to the right of the ~ we tell it what value to take.

We finally set a default value (TRUE) which in this case is just the original value.

```
# if column A has a value called "aa1", change it to "a1", else leave
cleaned_data$A <- case_when(
  cleaned_data$A == "aa1" ~ "a1",
  TRUE ~ cleaned_data$A
)

# check the new values
unique(cleaned_data$A)
```

```
## [1] "a2" "a1"
```

Fixing Remaining Problems

We can put multiple cases in `case_when()`! This way, it can be **more flexible and less repetitive** than other methods.

```r
cleaned_data$B <- case_when(
  cleaned_data$B == "bb1" ~ "b1",
  cleaned_data$B == "bb2" ~ "b2",
  TRUE ~ cleaned_data$B
)

# check new values
unique(cleaned_data$B)
```

```
## [1] "b2" "b1"
```

Now we only have B1 and B2! It looks like our data is nicely tidied up now.

Save our Cleaned Data

Finally, we did all that work with our data. To save repeating all these steps again, we can save it to an external file. We do this using 'write_csv()`.

Let's save our file in the data folder. This takes two arguments: (1) which R object will you save to file? (2) Where should you save it?

```
write_csv(
  cleaned_data,
  here("data", "cleaned_data.csv")
)
```

The nice thing is, if you made a mistake in one of your steps, you can just change that step and rerun your code.

If you change things by hand, who knows if you'll even detect the mistake, never mind how difficult it t might be to fix it!

Recap

We know…

- how to read data into R.
- how to know how our data is coded.
- how to inspect subsets of our data.
- how to identify unique elements of our data.
- how to convert and clean up data.
- how to perform case-specific operations using logic.