Managing Data in R

Data Checking in R

Glenn Williams

University of Sunderland

2021-10-27 (updated: 2021-10-27)

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 made**. Ask yourself:

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

Answering these questions makes you understand your data better, so you'll make fewer mistakes and do a better job of analysing it.

Reading in Data

In the data folder of this repository are files called **factorial_data_a.csv** and **factorial_data_b.csv**. If I'm working from this project folder I can read it in and combine the data as follows:

```
# read the data
data_a <- read_csv(here("data", "factorial_data_a.csv"))
data_b <- read_csv(here("data", "factorial_data_b.csv"))

# combine the data
combined_data <- bind_rows(data_a, data_b)

# print it
combined_data
```

```
## # A tibble: 2,400 x 7
##     row 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.
## # … with 2,396 more rows
```

Some Notes on the Data

What does the data look like? The `head()` function allows you to see the top of your data set.

```
head(combined_data)
```

```
## # A tibble: 6 x 7
##     row 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.
```

The function `tail()` does the opposite to `head()`. Try it out!

Understanding our Data

Let's assume we gave people a task where they had to read **sentences that imply 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).

We have the following columns:

- **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 the data is coded as expected.

Inspecting Data

- Our first step to understanding our 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 on its side**, so we can see how our data are stored and how they are coded.

```
glimpse(combined_data)
```

```
## Rows: 2,400
## Columns: 7
## $ row     <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
```

There's a few problems with our coding system here. Subject ID should probably be numeric, and we need consistent codes for A and B.

Inspecting Data

- We can get a summary of our data, albeit in a confusing printout, using `summary()`.
- Here, we'll reduce the summary only to our numeric columns for easier presentation (selecting columns, only if they are numeric).

```
summary(select_if(combined_data, is.numeric))
```

```
##       row             list_id        item_id             Y          
##  Min.   :   1.0   Min.   :1.00   Min.   : 1.00   Min.   :-200.0  
##  1st Qu.: 600.8   1st Qu.:1.75   1st Qu.:10.75   1st Qu.: 231.7  
##  Median :1200.5   Median :2.50   Median :20.50   Median : 251.6  
##  Mean   :1200.5   Mean   :2.50   Mean   :20.50   Mean   : 255.4  
##  3rd Qu.:1800.2   3rd Qu.:3.25   3rd Qu.:30.25   3rd Qu.: 270.6  
##  Max.   :2400.0   Max.   :4.00   Max.   :40.00   Max.   :4000.0  
##                                                  NA's   :45
```

Our Y outcome, reaction time, seems to have a minimum of a negative value. We need to get rid of this clear error.

Inspecting Codes

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.

- We can ask for only values from one column using dollar indexing; e.g. `data$column`.
- The `unique()` function tells us what unique observations we have in a column.

Which unique values do we have here?

```
unique(combined_data$A)
```

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

```
unique(combined_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 Problems

Set the values in A and B to lower case.

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

How does it look?

```
unique(combined_data$A)
```

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

```
unique(combined_data$B)
```

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

We've fixed the codes for variable A, but variable B still has some problems. Someone has mistakenly entered bb2 and bb1 as codes.

Let's change these!

Fixing Remaining Problems

We'll use the `case_when()` function again here.

If column B has the value "bb1" change it to "b1", otherwise if it has "bb2" change it to "b2", else just leave values as they are.

```
combined_data <- mutate(
  combined_data,
  B = case_when(
    B == "bb1" ~ "b1",
    B == "bb2" ~ "b2",
    TRUE ~ B
  )
)
```

How do the values look now?

```
unique(combined_data$B)
```

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

Nice and consistent!

Identifying Remaining Problems

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

```
unique(combined_data$subj_id)
```

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

We have a mix of integers, and the two values "one" and "last" as IDs. We need to make this consistent.

Converting Values Conditionally

Let's change the "one" and "last" values to more sensible values. Notice we have to stick with characters for now so the column has the same data types in it.

```
combined_data$subj_id <- case_when(
  combined_data$subj_id == "one" ~ "1",
  combined_data$subj_id == "twenty-eight" ~ "28",
  combined_data$subj_id == "last" ~ "60",
  TRUE ~ combined_data$subj_id
)
```

How have the values changed?

```
unique(combined_data$subj_id)
```

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

Nicely, the numbers now go from 1 to 60! But they're still characters.

Fixing Remaining Problems

We can convert between data types using the `as.` functions. We have e.g. `as.character()`, `as.factor()`, and `as.numeric()`. We'll use `as.numeric()` to make these strings into numeric values now they're consistently coded.

```
combined_data$subj_id <- as.numeric(combined_data$subj_id)
```

How do the values look? Check the data again. All good!

```
glimpse(combined_data)
```

```
## Rows: 2,400
## Columns: 7
## $ row     <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
## $ subj_id <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
## $ 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
```

Fixing Remaining Problems

Finally, we can't have negative reaction times. Let's remove any reaction times below 100ms under the assumption they're far too quick for this task.

```
# filter only to RTs above 100ms
combined_data <- filter(combined_data, Y > 100)

# inspect it
combined_data
```

```
## # A tibble: 2,349 x 7
##     row subj_id list_id item_id A     B          Y
##   <dbl>   <dbl>   <dbl>   <dbl> <chr> <chr> <dbl>
## 1     1       1       4       1 a2    b2     234.
## 2     2       1       4       2 a2    b2     295.
## 3     3       1       4       3 a2    b2     285.
## 4     4       1       4       4 a2    b2     258.
## # … with 2,345 more rows
```

All done!

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? What will the data be called?

```
write_csv(
  combined_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 might be to fix it!

Recap

We know...

- how to read data into R.
- how to find out how our data are coded.
- how to identify unique elements of our data.
- how to get a summary of our data.
- how to convert and clean up data.
- how to perform case-specific operations using conditional logic.