Managing Data in R

Manipulating Data in R

Glenn Williams

University of Sunderland

2020-09-29 (updated: 2020-10-14)

Loading our Cleaned Data

Let's load up our cleaned data from the prior mini-lecture.

Remember, `read_csv()` takes one argument: where is your data!

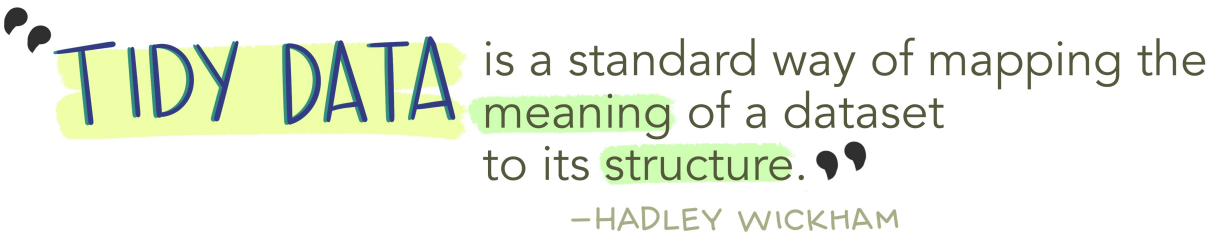
```
cleaned_data <- read_csv(here("data", "cleaned_data.csv"))
```

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

Nicely, you can see the hard work we did! The ID columns are all numbers now, and only the A and B columns are characters.

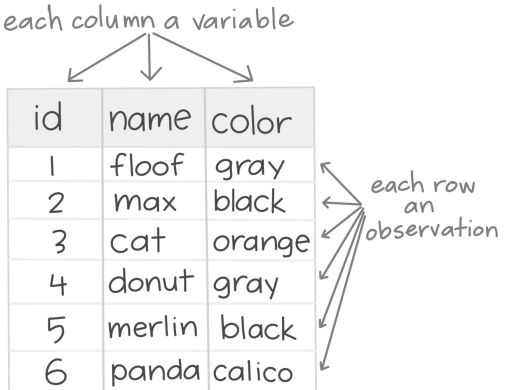
Tidy Data

Our data is in a tidy format.



In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement

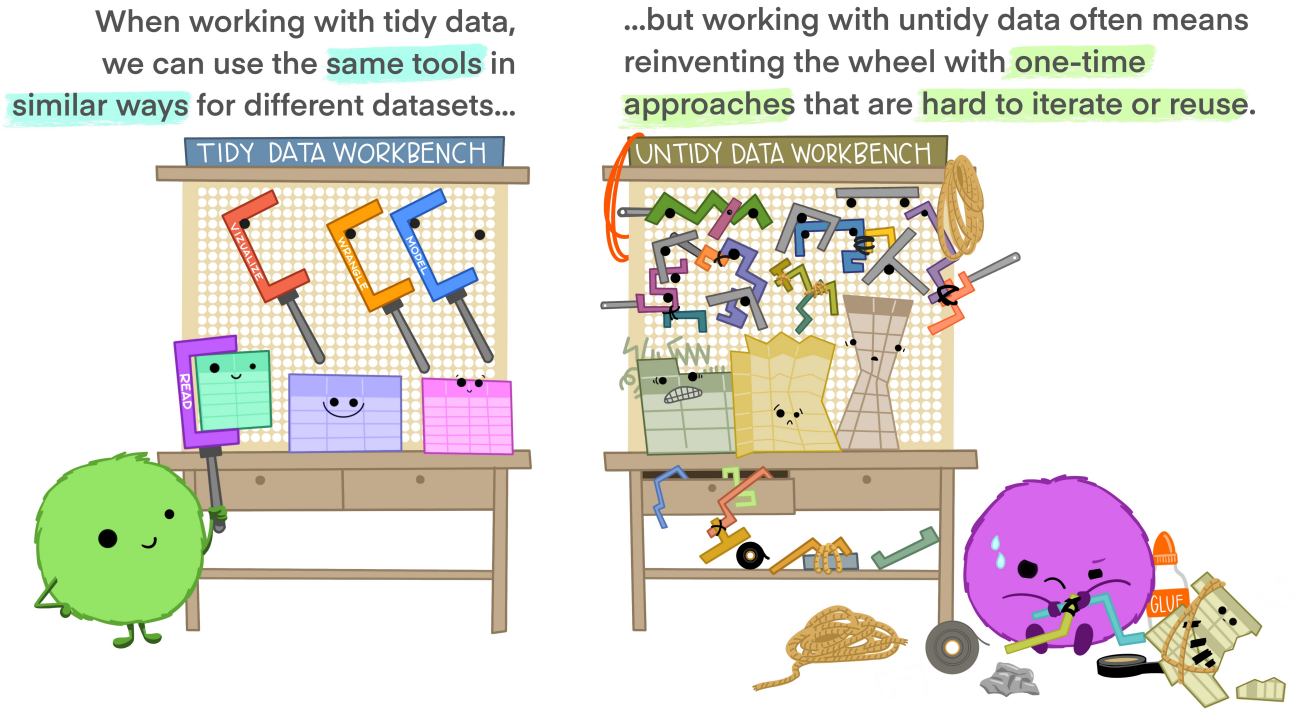


Wickham, H. (2014). Tidy Data. Journal of Statistical Software *59* (10). DOI: 10.18637/jss.v059.i10

Illustrations from the Openscapes blog Tidy Data for reproducibility, efficiency, and collaboration by Julia Lowndes and Allison Horst

Why Ensure Tidy Data?

This makes cleaning, summarising, plotting, and modelling easier!



Illustrations from the Openscapes blog Tidy Data for reproducibility, efficiency, and collaboration by Julia Lowndes and Allison Horst

Common Data Processing Tasks

The most common data manipulation problems can be handled effectively by `dplyr` in the tidyverse:

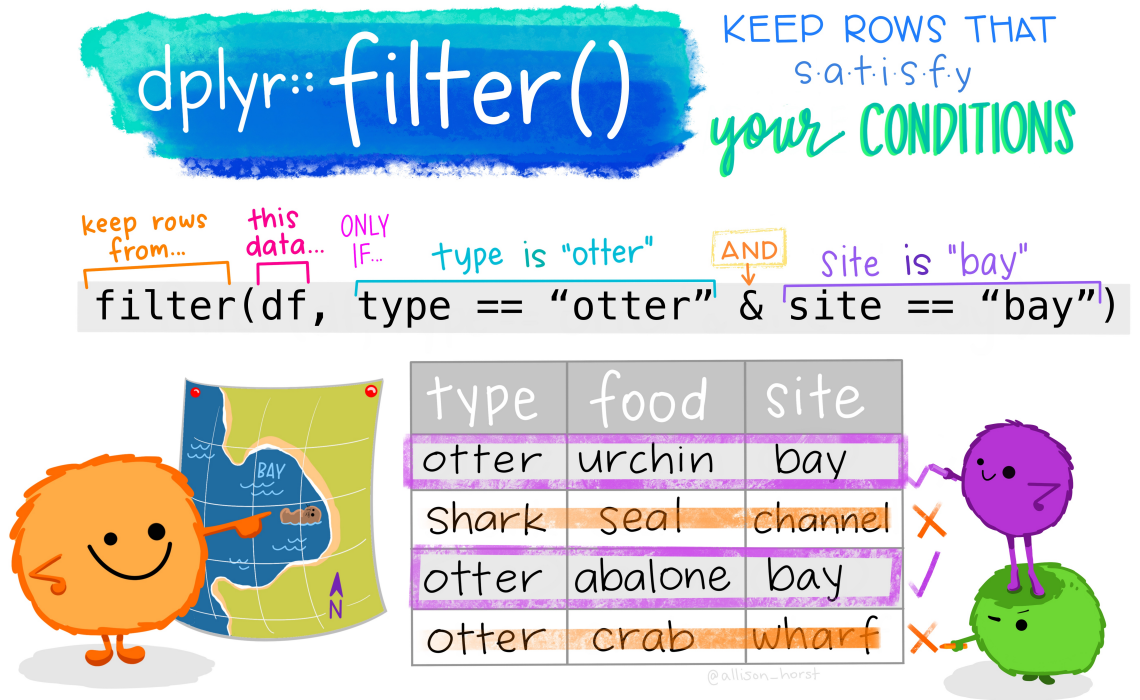
One Table:

- `filter()`: filters our data to keep observations that only meet given conditions.
- `select()`: selects a subset of columns in our table.
- `mutate()`: changes our data in some way by the instructions you provide.

Multiple Tables:

- `bind_rows()`: adds rows from one data frame to another (if they have the same columns).
- `bind_cols()`: adds columns to one data frame from another (if they have the same number of rows).

Filter



Artwork by @allison_horst

Filter

Let's say we want to keep only observations from our first participant. This might be useful for exploring data individually.

```
filter(cleaned_data, subj_id == 1)
```

```
## # A tibble: 40 x 6
##   subj_id list_id item_id A     B         Y
##     <dbl>   <dbl>   <dbl> <chr> <chr> <dbl>
## 1       1       4       1 a2    b2     234.
## 2       1       4       2 a2    b2     295.
## 3       1       4       3 a2    b2     285.
## 4       1       4       4 a2    b2     258.
## # … with 36 more rows
```

We have only 40 observations now, all from subject 1.

Combining Filter Conditions

Maybe we want to check if subject 1 has any missing data in the variable Y. How might we do this?

We could apply two separate filters, or combine our filter conditions using logical operations.

```
filter(cleaned_data, subj_id == 1 & is.na(Y))
```

```
## # A tibble: 4 x 6
##   subj_id list_id item_id A     B          Y
##     <dbl>   <dbl>   <dbl> <chr> <chr> <dbl>
## 1       1       4      17 a1    b1       NA
## 2       1       4      24 a1    b2       NA
## 3       1       4      32 a2    b1       NA
## 4       1       4      33 a2    b1       NA
```

Note that NAs are special in R, so we have to use is.na() which checks for any NA values explicitly, rather than Y == NA. (Confusing, I know...)

It looks like subject 1 has four missing responses!

Select

Select works by defining the names of columns you'd like to keep, and in what order!

Artwork by @allison_horst

Select

Imagine we only want to keep the subject_ids.

```
select(cleaned_data, subj_id)
```

```
## # A tibble: 2,400 x 1
##   subj_id
##     <dbl>
## 1       1
## 2       1
## 3       1
## 4       1
## # … with 2,396 more rows
```

Select

Imagine we decide that we no longer care about the list_id. We simply put a minus before the column name.

```
select(cleaned_data, -list_id)
```

```
## # A tibble: 2,400 x 5
##   subj_id item_id A     B          Y
##     <dbl>   <dbl> <chr> <chr> <dbl>
## 1       1       1 a2    b2     234.
## 2       1       2 a2    b2     295.
## 3       1       3 a2    b2     285.
## 4       1       4 a2    b2     258.
## # … with 2,396 more rows
```

Select

Maybe we'd like to reorganise our columns so condition comes first and everything else after. We have to tell select we want A and B, and then everything else using `everything()`; a dplyr helper function

```
select(cleaned_data, A, B, everything())
```

```
## # A tibble: 2,400 x 6
##   A     B     subj_id list_id item_id     Y
##   <chr> <chr>   <dbl>   <dbl>   <dbl> <dbl>
## 1 a2    b2          1       4       1  234.
## 2 a2    b2          1       4       2  295.
## 3 a2    b2          1       4       3  285.
## 4 a2    b2          1       4       4  258.
## # … with 2,396 more rows
```

Muatate

Artwork by @allison_horst

Mutate

With mutate, we can change our data in place. This means we can change the data within columns, or create new columns of data entirely.

Let's create a new column. We might want to make a column which combines the strings of columns A and B. We'll use the paste function to do this, telling R to paste together strings from cells in columns A and B, and include a + sign between them.

```
mutate(cleaned_data, AB = paste(A, B, sep = "+"))
```

```
## # A tibble: 2,400 x 7
##   subj_id list_id item_id A     B         Y AB
##     <dbl>   <dbl>   <dbl> <chr> <chr> <dbl> <chr>
## 1       1       4       1 a2    b2     234. a2+b2
## 2       1       4       2 a2    b2     295. a2+b2
## 3       1       4       3 a2    b2     285. a2+b2
## 4       1       4       4 a2    b2     258. a2+b2
## # … with 2,396 more rows
```

Mutate

Perhaps instead we want to change the data in a column in place? Let's multiply the scores for Y by 100.

```
mutate(cleaned_data, Y = Y*100)
```

```
## # A tibble: 2,400 x 6
##   subj_id list_id item_id A     B           Y
##     <dbl>   <dbl>   <dbl> <chr> <chr>   <dbl>
## 1       1       4       1 a2    b2    23351.
## 2       1       4       2 a2    b2    29467.
## 3       1       4       3 a2    b2    28462.
## 4       1       4       4 a2    b2    25798.
## # … with 2,396 more rows
```

Mutate

Or we can use cases instead to change only specific values.

```
mutate(
  cleaned_data,
  item_id = case_when(
    item_id == 1 ~ 20,
    TRUE ~ item_id
  )
)
```

```
## # A tibble: 2,400 x 6
##   subj_id list_id item_id A     B         Y
##     <dbl>   <dbl>   <dbl> <chr> <chr> <dbl>
## 1       1       4      20 a2    b2     234.
## 2       1       4       2 a2    b2     295.
## 3       1       4       3 a2    b2     285.
## 4       1       4       4 a2    b2     258.
## # … with 2,396 more rows
```

Bind Rows

Let's imagine we had our data stored in two separate tables. How might we join the rows of the tables together?

Making this easy, we'll just take out the first two rows of our data and all their columns.

We'll then join the rows together using `bind_rows()`.

```
# get our data
row_one <- cleaned_data[1, ]
row_two <- cleaned_data[2, ]

# bind it together
bind_rows(row_one, row_two)
```

```
## # A tibble: 2 x 6
##   subj_id list_id item_id A     B         Y
##     <dbl>   <dbl>   <dbl> <chr> <chr> <dbl>
## 1       1       4       1 a2    b2     234.
## 2       1       4       2 a2    b2     295.
```

Bind Cols

We have a similar process when we bind columns.

First, we'll extract the columns A and B, both only having the first 4 rows of our data each.

Then to show how the function works, we'll bind these objects together using `bind_cols()`.

```
# get our data
col_a <- cleaned_data[1:4, "A"]
col_b <- cleaned_data[1:4, "B"]

# bind it together
bind_cols(col_a, col_b)
```

```
## # A tibble: 4 x 2
##   A     B
##   <chr> <chr>
## 1 a2    b2
## 2 a2    b2
## 3 a2    b2
## 4 a2    b2
```

Recap

We know...

- What tidy data is, and why we want our data to be in this format.
- How to manipulate one-table data for cleaning and preparation for analysis/graphing, including:
    - How to filter observations out of our data set.
    - How to select particular columns in our data set.
    - How to mutate or modify existing data and even create new columns of data.
    - How to bind rows and columns together from multiple tables.