Day 4, Part 2: Tidying and Cleaning Data

Brennan Terhune-Cotter and Matt Dye

https://github.com/brennangitsit/2023_IAM3_R

Agenda

1. Missing data (NA)
2. Untidy to tidy data

Missing data (NA)

- NA is a very tricky data type!
- NA = *unknown*
- NAs are contagious

```
1  NA == 50
```

```
[1] NA
```

```
1  NA > 7
```

```
[1] NA
```

```
1  NA == NA
```

```
[1] NA
```

Missing data (NA)

c1	c2
1	3
2	NA
3	5
4	NA
5	NA
6	9

We can't use regular logical expressions to find or remove NA values:

```
1 df1 %>% filter(c2 == NA)
```

```
[1] c1 c2
<0 rows> (or 0-length row.names)
```

We have to use specialized functions such as `is.na()` or its opposite, `!is.na()`

```
1 df1 %>% filter(!is.na(c2))
```

```
  c1 c2
1  1  3
2  3  5
3  6  9
```

Other functions for missing data

- `na.omit()`: This function removes the **rows** with `NA` values from a data frame or a vector.
- `complete.cases()`: This function returns a logical vector indicating which cases/rows have no missing values.
- `sum(is.na())`: While not a specific function, this combination is often used to count the number of `NA`s in a vector or a column of a data frame.
- `na.rm = TRUE` is an argument you will use very often to ensure you're removing missing values from visualizations or tables.

What is a (Tidy) Data Frame?

Why Tidy Data?

- Hadley tells us:
  - "There's a general advantage to picking one consistent way of storing data. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity."
  - "There's a specific advantage to placing variables in columns because it allows R's **vectorized nature** to shine … most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural."

Vectors

- Vectors are very simple but can get very complicated. They form the core of a lot of things you can do with R
- Vectors are simply a *list* of *items that are the same type*
- Often, you will want to work with columns in a dataframe as vectors

> **variables in a dataframe == columns == vectors**

Lists

- A type of vector which is more flexible
  - A “collection” of objects or variables
  - Vectors are useful for simplicity; lists are useful for many other things
- Elements in a list can contain any type of R object
  - Elements can be different types and different structures
  - Including other lists! (i.e. *nested list*)
  - Elements can be named

```
1 listylist <- list(3, "42", "hello!", c(3,4,5))
2 listylist[[4]]
```

```
[1] 3 4 5
```

a list of vectors == a data frame!!! **

```r
people <- data.frame(
  name = c("Alice", "Spew", "Charlemagne", "Kay", "Mackenzie", "Spirulina", "Jason"),
  age = c(25, 16, 42, 18, 3, 16, 20),
  humor_score = c(7, 14, 16, 9, 1, 20, 11),
  humor_category = c("Bad", "Good", "Good", "Bad", "Terrible", "Too Good", "Good")
)
kable(people)
```

name	age	humor_score	humor_category
Alice	25	7	Bad
Spew	16	14	Good
Charlemagne	42	16	Good
Kay	18	9	Bad
Mackenzie	3	1	Terrible
Spirulina	16	20	Too Good
Jason	20	11	Good

**(there are some other limitations, like the vectors have to be the same length, but this is the general concept 🤷)

Untidy to Tidy Data

Your Untidy Data

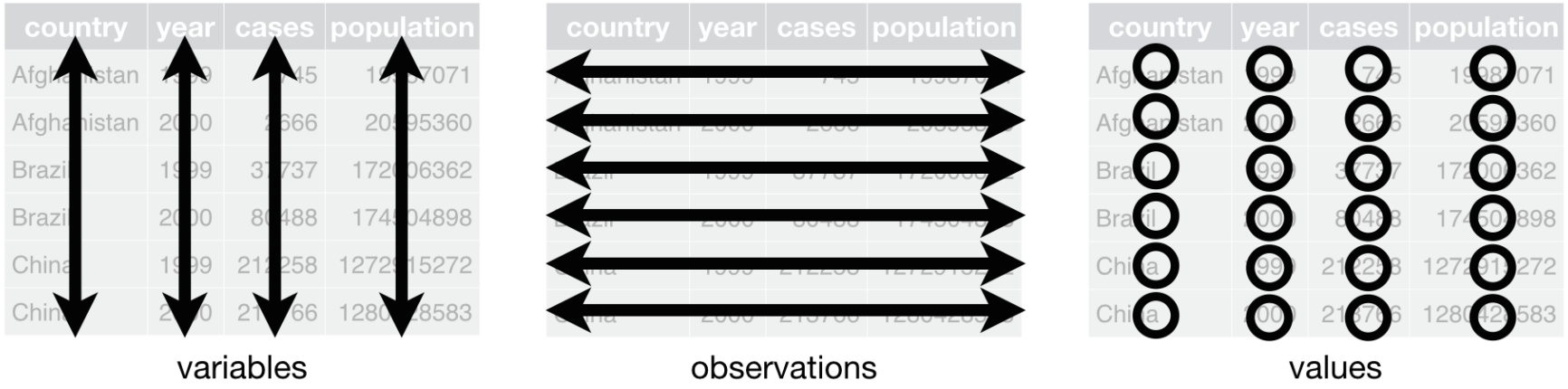
You have a dataframe that contains test scores collected at two time points: test_t1 and test_t2.

subjid	test1_t1	test1_t2
iam3_001	19	20
iam3_002	18	18
iam3_003	19	17
iam3_004	17	22
iam3_005	19	21
iam3_006	17	18
iam3_007	15	19
iam3_008	21	22
iam3_009	20	24
iam3_010	18	19

Flash back to Data Visualization...

- Before we can correctly visualize our test data, we need it in tidy format.
- Why?
  - Because with untidy data, ggplot doesn't know how "test_t1" and "test_t2" are related
  - Tidying the data allowed us to give ggplot appropriate x and y variables

Long/Tidy vs. Short/Untidy Data



In tidy data:

1. Every column is a variable.
2. Every row is an observation.
3. Every cell is a single value.

pivot_longer()

pivot_longer() accepts the following key arguments:

1. **data** [dataframe to pivot]
2. **cols** [input columns to pivot]
3. **names_to** [name of new column with variable names]
4. **names_prefix** [prefix of input columns that will be removed]
5. **values_to** [name of new column with values]

Check out this data!

subjid	test1_t1	test1_t2
iam3_001	19	20
iam3_002	18	18
iam3_003	19	17
iam3_004	17	22
iam3_005	19	21
iam3_006	17	18
iam3_007	15	19
iam3_008	21	22
iam3_009	20	24
iam3_010	18	19

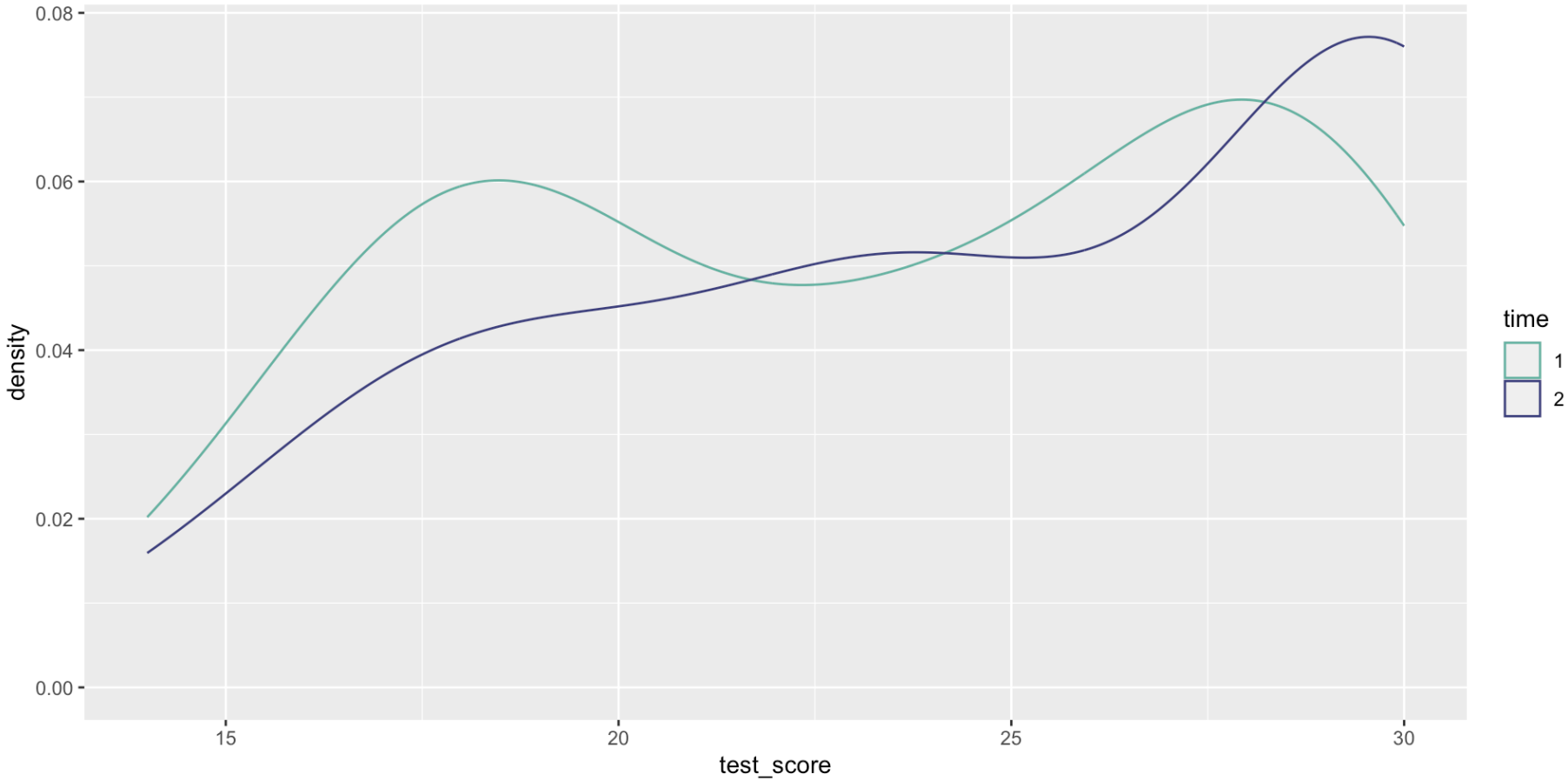
Let's tidy this data!

```
tidy_lang_data_simple <- pivot_longer(lang_data_simple,
                          test1_t1:test1_t2,
                          names_to = "time",
                          names_prefix = "test1_t",
                          values_to = "test_score")
tidy_lang_data_simple %>% head(15) %>% kable()
```

subjid	time	test_score
iam3_001	1	19
iam3_001	2	20
iam3_002	1	18
iam3_002	2	18
iam3_003	1	19
iam3_003	2	17
iam3_004	1	17
iam3_004	2	22
iam3_005	1	19
iam3_005	2	21

iam3_006	1	17
iam3_006	2	18
iam3_007	1	15
iam3_007	2	19
iam3_008	1	21

Now we can visualize our data



More complex data

This was a (relatively) simple scenario. Sometimes our data looks more complex.

For example, we might have scores for more than one language test, and also record the age of the participant when they took the test.

subjid	age_t1	age_t2	test1_t1	test2_t1	test3_t1	test1_t2	test2_
iam3_001	6	7	19	35	4	20	
iam3_002	5	7	18	41	2	18	
iam3_003	7	8	19	36	6	17	
iam3_004	8	10	17	45	11	22	
iam3_005	9	11	19	32	6	21	
iam3_006	8	10	17	35	8	18	
iam3_007	9	11	15	38	10	19	
iam3_008	10	12	21	40	11	22	
iam3_009	9	10	20	31	3	24	
iam3_010	12	14	18	30	9	19	

Tidying complex data

```
tidy_lang_data_complex <- pivot_longer(lang_data_complex,
                          cols = !subjid,
                          names_to = c(".value", "time"),
                          names_sep = "_")
tidy_lang_data_complex %>% head(15) %>% kable()
```

subjid	time	age	test1	test2	test3
iam3_001	t1	6	19	35	4
iam3_001	t2	7	20	35	5
iam3_002	t1	5	18	41	2
iam3_002	t2	7	18	40	3
iam3_003	t1	7	19	36	6
iam3_003	t2	8	17	35	8
iam3_004	t1	8	17	45	11
iam3_004	t2	10	22	47	12
iam3_005	t1	9	19	32	6
iam3_005	t2	11	21	32	8
iam3_006	t1	8	17	35	8

iam3_006	t2	10	18	34	9
iam3_007	t1	9	15	38	10
iam3_007	t2	11	19	40	11
iam3_008	t1	10	21	40	11

Bonus wrangling!

Now we do a little more wrangling, because we know that time is an ordinal variable indicating the wave of data collection.

```
# A tibble: 15 × 6
   subjid   time    age test1 test2 test3
   <chr>    <fct> <int> <int> <int> <int>
 1 iam3_001 1         6    19    35     4
 2 iam3_001 2         7    20    35     5
 3 iam3_002 1         5    18    41     2
 4 iam3_002 2         7    18    40     3
 5 iam3_003 1         7    19    36     6
 6 iam3_003 2         8    17    35     8
 7 iam3_004 1         8    17    45    11
 8 iam3_004 2        10    22    47    12
 9 iam3_005 1         9    19    32     6
10 iam3_005 2        11    21    32     8
11 iam3_006 1         8    17    35     8
12 iam3_006 2        10    18    34     9
13 iam3_007 1         9    15    38    10
14 iam3_007 2        11    19    40    11
15 iam3_008 1        10    21    40    11
```

Our Tidy Data!

Now. we have “tidy” data because:

- Every column is a variable: test1_t1 was not a variable because it indicates the test score at a specific point in time, whereas test1 is a variable equal to the score observed on a particular test
- Every row contains a single “observation”, where an observation is data collected from a single individual at a single point in time
- Each cell contains a single value

```
# A tibble: 15 × 6
   subjid   time    age test1 test2 test3
   <chr>    <fct> <int> <int> <int> <int>
 1 iam3_001 1         6    19    35     4
 2 iam3_001 2         7    20    35     5
 3 iam3_002 1         5    18    41     2
 4 iam3_002 2         7    18    40     3
 5 iam3_003 1         7    19    36     6
 6 iam3_003 2         8    17    35     8
 7 iam3_004 1         8    17    45    11
 8 iam3_004 2        10    22    47    12
 9 iam3_005 1         9    19    32     6
10 iam3_005 2        11    21    32     8
11 iam3_006 1         8    17    35     8
12 iam3_006 2        10    18    34     9
13 iam3_007 1         9    15    38    10
14 iam3_007 2        11    19    40    11
```

Next...

Either:

- Try tidying some data using **15_tidying.Rmd**

Or:

- Treat this as an OYOLab and see if your own data needs some tidying!