# Day 4, Part 2:

# Tidying and Cleaning Data

Brennan Terhune-Cotter and Matt Dye

## 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

```
NA == 50
```

```
[1] NA
```

```
NA > 7
```

```
[1] NA
```

```
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:

```
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()`

```
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.

# Untidy to Tidy Data

## 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.

## Your Untidy Data

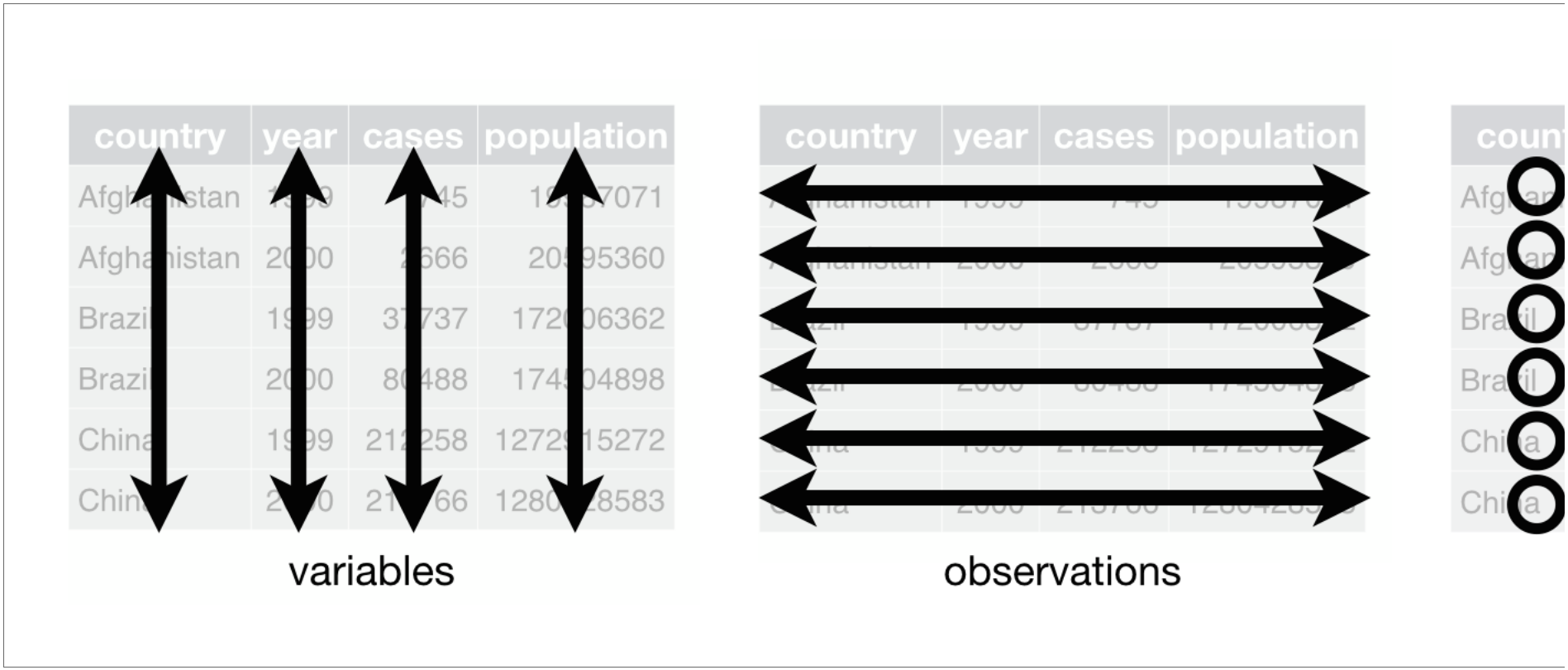
You have a dataframe that contains test scores collected at two time points: test_t1 and test_t2.

| 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 an appropriate x-axis variable (time) and y-axis variable (test score).

## 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!

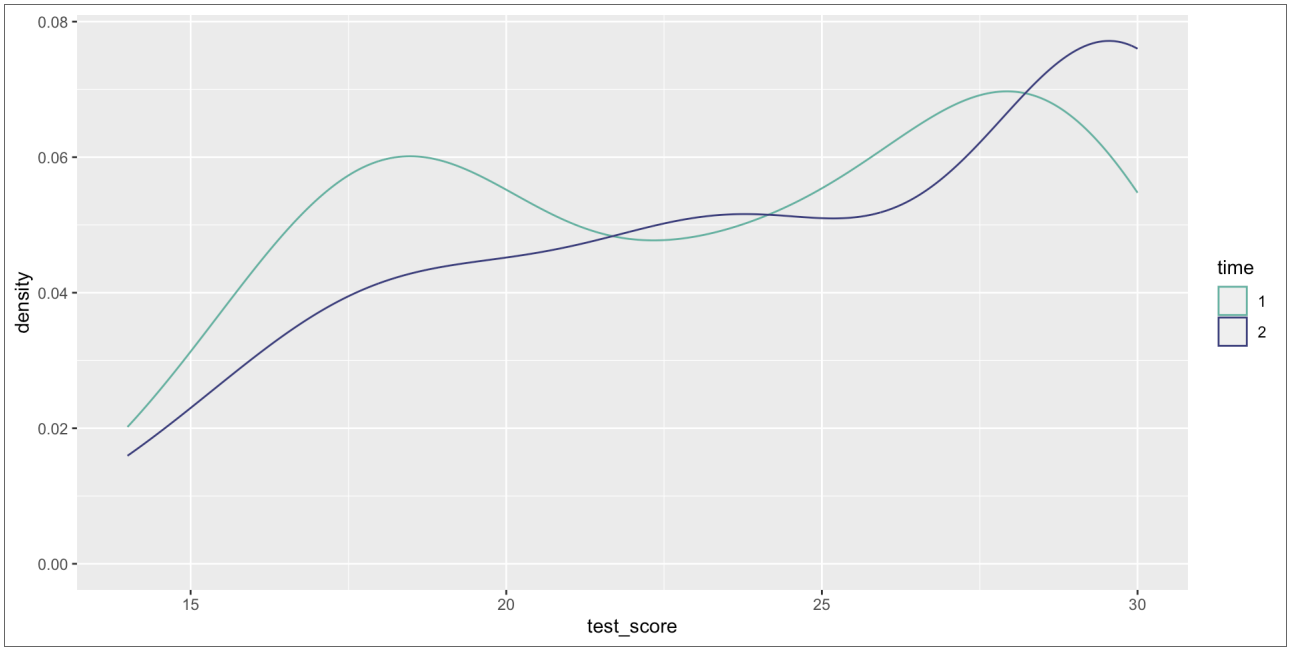
| subjid | test1_t1 | test1_t2 |
|---|---|---|
| iam3_001 | 19 | 20 |
| iam3_002 | 18 | 18 |
| iam3_003 | 19 | 17 |
| iam3_004 | 17 | 22 |
| iam3_005 | 19 | 21 |
| iam3_006 | 17 | 18 |
| iam3_007 | 15 | 19 |
| iam3_008 | 21 | 22 |
| iam3_009 | 20 | 24 |
| iam3_010 | 18 | 19 |

# 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_t2 | test3_t2 |
|---|---|---|---|---|---|---|---|---|
| iam3_001 | 6 | 7 | 19 | 35 | 4 | 20 | 35 | 5 |
| iam3_002 | 5 | 7 | 18 | 41 | 2 | 18 | 40 | 3 |
| iam3_003 | 7 | 8 | 19 | 36 | 6 | 17 | 35 | 8 |
| iam3_004 | 8 | 10 | 17 | 45 | 11 | 22 | 47 | 12 |
| iam3_005 | 9 | 11 | 19 | 32 | 6 | 21 | 32 | 8 |
| iam3_006 | 8 | 10 | 17 | 35 | 8 | 18 | 34 | 9 |
| iam3_007 | 9 | 11 | 15 | 38 | 10 | 19 | 40 | 11 |
| iam3_008 | 10 | 12 | 21 | 40 | 11 | 22 | 39 | 13 |
| iam3_009 | 9 | 10 | 20 | 31 | 3 | 24 | 30 | 4 |
| iam3_010 | 12 | 14 | 18 | 30 | 9 | 19 | 28 | 10 |

## 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)
```

```
# A tibble: 15 × 6
   subjid   time    age test1 test2 test3
   <chr>    <chr> <int> <int> <int> <int>
 1 iam3_001 t1        6    19    35     4
 2 iam3_001 t2        7    20    35     5
 3 iam3_002 t1        5    18    41     2
 4 iam3_002 t2        7    18    40     3
 5 iam3_003 t1        7    19    36     6
 6 iam3_003 t2        8    17    35     8
 7 iam3_004 t1        8    17    45    11
 8 iam3_004 t2       10    22    47    12
 9 iam3_005 t1        9    19    32     6
10 iam3_005 t2       11    21    32     8
11 iam3_006 t1        8    17    35     8
12 iam3_006 t2       10    18    34     9
13 iam3_007 t1        9    15    38    10
14 iam3_007 t2       11    19    40    11
15 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
15 iam3_008 1        10    21    40    11
```

## Running a Paired T-Test

If we wanted to compare performance on test1 and times 1 and 2 (to see if scores change), then we should run a "paired" t-test that takes into account the fact that the scores at time 1 and time 2 were obtained from the same individuals:

```
ttest_test1 <- t.test(test1 ~ time, data = tidy_lang_data_complex, paired = TRUE)
ttest_test1
```

**Error**

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