Data Wrangling with R: Day 2

Relational data wrangling, starring janitor and broom

Presented by Emi Tanaka

Department of Econometrics and Business Statistics

2nd December 2020 @ Statistical Society of Australia | Zoom
Relational data
Do you remember what the definition of a **tidy data** is?

**Definition of a tidy data**
- Each variable must have its own column
- Each observation must have its own row
- Each value must have its own cell
Observational unit

- In the original definition in Wickham (2014), which was a statistical version of the definition in Codd (1990), the third point originally was actually:

> Each type of observational unit forms a table.

- What is an **observational unit**?
- It's the unit in which measurement is made, e.g. if measure height of a person then the person is the observational unit

The data on the left shows the height of a barley variety at a given year in Norway experiment.

The temperature at six different time points of the growth barley was recorded.

What's the observational unit here?

Yes. there are two observational unit here: the barley and the environment at six different time points per year.
- Originally the data were in a separate table
- Notice before that the height measurements were duplicated
- While it's tidier to have it separated like this, you may need to join the data for downstream analysis

```markdown
<table>
<thead>
<tr>
<th>#</th>
<th>year</th>
<th>period</th>
<th>temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1974</td>
<td>T1</td>
<td>8.36</td>
</tr>
<tr>
<td>2</td>
<td>1974</td>
<td>T2</td>
<td>9.60</td>
</tr>
<tr>
<td>3</td>
<td>1974</td>
<td>T3</td>
<td>12.11</td>
</tr>
<tr>
<td>4</td>
<td>1974</td>
<td>T4</td>
<td>12.13</td>
</tr>
<tr>
<td>5</td>
<td>1974</td>
<td>T5</td>
<td>18.43</td>
</tr>
<tr>
<td>6</td>
<td>1974</td>
<td>T6</td>
<td>13.75</td>
</tr>
<tr>
<td>7</td>
<td>1975</td>
<td>T1</td>
<td>7.66</td>
</tr>
<tr>
<td>8</td>
<td>1975</td>
<td>T2</td>
<td>11.66</td>
</tr>
<tr>
<td>9</td>
<td>1975</td>
<td>T3</td>
<td>10.06</td>
</tr>
</tbody>
</table>
```

```markdown
<table>
<thead>
<tr>
<th>#</th>
<th>year</th>
<th>variety</th>
<th>height</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1974</td>
<td>G01</td>
<td>81.0</td>
</tr>
<tr>
<td>2</td>
<td>1975</td>
<td>G01</td>
<td>67.3</td>
</tr>
<tr>
<td>3</td>
<td>1976</td>
<td>G01</td>
<td>71.5</td>
</tr>
<tr>
<td>4</td>
<td>1977</td>
<td>G01</td>
<td>64.3</td>
</tr>
<tr>
<td>5</td>
<td>1978</td>
<td>G01</td>
<td>55.8</td>
</tr>
<tr>
<td>6</td>
<td>1979</td>
<td>G01</td>
<td>84.9</td>
</tr>
<tr>
<td>7</td>
<td>1980</td>
<td>G01</td>
<td>86.2</td>
</tr>
<tr>
<td>8</td>
<td>1981</td>
<td>G01</td>
<td>88.0</td>
</tr>
<tr>
<td>9</td>
<td>1982</td>
<td>G01</td>
<td>72.0</td>
</tr>
</tbody>
</table>
```
So, how do you join two related data?

Yes, you can only join the table if each table has columns that you can join by
Joining datasets with dplyr

There are many ways to do so
**Inner join**

`inner_join(x, y)`

- All rows from `x` where there are matching values in `y`, and all columns from `x` and `y`. 

GIF credit: Garrick Aden-Buie
If there are multiple matches between x and y, all combinations of the matches are returned.

All rows from x, and all columns from x and y. Rows in x with no match in y will have NA values in the new columns.
All rows from y, and all columns from x and y. Rows in y with no match in x will have NA values in the new columns.
Full join

`full_join(x, y)`

- All rows and all columns from both `x` and `y`. Where there are not matching values, returns `NA` for the one missing.
Semi join

- All rows from x where there are matching values in y, keeping just columns from x.

semi_join(x, y)
Anti join

anti_join(x, y)

- All rows from x where there are not matching values in y, keeping just columns from x.
Set Operations with Relational Data
Union

union(x, y)

- All unique rows from x and y.

GIF credit: Garrick Aden-Buie
Union all

union_all(x, y)

<table>
<thead>
<tr>
<th></th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
</tbody>
</table>

- All rows from x and y, keeping duplicates.
**Intersection**

```
intersect(x, y)
```

- Common rows in both x and y, keeping just unique rows.
Set difference

\( \text{setdiff}(x, y) \)

- All rows from \( x \) which are not also rows in \( y \), keeping just unique rows.
setdiff(y, x)

- All rows from y which are not also rows in x, keeping just unique rows.
Joining by multiple columns

x

## # A tibble: 3 x 3
##  year site  value
##  <dbl> <chr> <dbl>
## 1 2010 A     1
## 2 2010 B     3
## 3 2011 B     2

y

## # A tibble: 2 x 4
##  year loc   value  resp
##  <dbl> <chr> <dbl> <dbl>
## 1 2010 A     1  5.4
## 2 2010 B     4  3

x %>%
left_join(y, 
  by = c("year", "site" = "loc"),
  suffix = c("_x", "_y"))

## # A tibble: 3 x 5
##  year site  value_x value_y  resp
##  <dbl> <chr>   <dbl>   <dbl> <dbl>
## 1 2010 A           1       1   5.4
## 2 2010 B           3       4   3
## 3 2011 B           2      NA  NA
Nesting data with tidyR
Nest data

df %>% as_tibble()

```r
# # A tibble: 810 x 5
# #   year variety height period temperature
# #    <int> <fct>    <dbl> <chr>    <chr>
# #  1  1974 G01       81   T1      1
# #  2  1974 G01       81   T2      1
# #  3  1974 G01       81   T3      1
# #  4  1974 G01       81   T4      1
# #  5  1974 G01       81   T5      1
# #  6  1974 G01       81   T6      1
# #  7  1975 G01       67.3 T1     1
# #  8  1975 G01       67.3 T2     1
# #  9  1975 G01       67.3 T3     1
# # 10  1975 G01       67.3 T4     1
# # ... with 800 more rows
```
df %>%

```r
nest(weather = period:temperature)
```

```r
# # A tibble: 135 x 4
# #   year variety height weather
# #    <int> <fct>    <dbl> <list>
# #  1  1974 G01       81   <tibble [6 x
# #  2  1975 G01       67.3 <tibble [6 x
# #  3  1976 G01       71.5 <tibble [6 x
# #  4  1977 G01       64.3 <tibble [6 x
# #  5  1978 G01       55.8 <tibble [6 x
# #  6  1979 G01       84.9 <tibble [6 x
# #  7  1980 G01       86.2 <tibble [6 x
# #  8  1981 G01       88    <tibble [6 x
# #  9  1982 G01       72    <tibble [6 x
# # 10  1974 G02       72.3 <tibble [6 x
# # ... with 125 more rows
```
```r
df %>%
  nest(weather = period:temperature) %>%
  rowwise() %>%
  mutate(avg_temp = mean(weather$temperature))
```

```
# # A tibble: 135 x 5
# # Rowwise:
# #     year variety height weather          avg_temp
# #    <int> <fct>    <dbl> <list>              <dbl>
# 1  1974 G01       81   <tibble [6 × 2]>     12.4
# 2  1975 G01       67.3 <tibble [6 × 2]>     13.4
# 3  1976 G01       71.5 <tibble [6 × 2]>     14.6
# 4  1977 G01       64.3 <tibble [6 × 2]>     14.7
# 5  1978 G01       55.8 <tibble [6 × 2]>     13.8
# 6  1979 G01       84.9 <tibble [6 × 2]>     13.8
# 7  1980 G01       86.2 <tibble [6 × 2]>     14.5
```
Using nested data: group_by

- rowwise is different to using `group_by` even if `group_by` refers to each row

```r
df %>%
  nest(weather = period:temperature) %>%
  group_by(year, variety) %>%
  mutate(avg_temp = mean(weather[[1]]$temperature))
```

```r
# A tibble: 135 x 5
# Groups:   year, variety 

#  year variety height weather          avg_temp
#  <int> <fct>    <dbl> <list>              <dbl>
# 1  1974 G01       81   <tibble [6 × 2]>     12.4
# 2  1975 G01       67.3 <tibble [6 × 2]>     13.4
# 3  1976 G01       71.5 <tibble [6 × 2]>     14.6
# 4  1977 G01       64.3 <tibble [6 × 2]>     14.7
# 5  1978 G01       55.8 <tibble [6 × 2]>     13.8
```
Tidy model output with broom
Model outputs are generally messy

```r
fit <- lm(speed ~ dist, data = cars)
summary(fit)
```

```r
## Call:
## lm(formula = speed ~ dist, data = cars)
#### Residuals:
##     Min      1Q  Median      3Q     Max
##-7.5293 -2.1550  0.3615  2.4377  6.4179
#### Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept) 8.28391   0.87438   9.474 1.44e-12 ***
## dist       0.16557   0.01749   9.464 1.49e-12 ***
##---
```
```r
fit <- lm(speed ~ dist, data = cars)
tidy(fit)
```
```r
tidy(fit)
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)    8.28     0.874       9.47 1.44e-12
## 2 dist           0.166    0.0175      9.46 1.49e-12

```r
glance(fit)
```
```r
glance(fit)
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.651         0.644  3.16      89.6 1.49e-12     1  -127.  261.  267.
## # … with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```
```r
fit %>%
  augment()
```
Dealing with non-syntatic names with janitor
Non-syntactic variable names

• E.g. "loc@nsw", "Frog ID" and "_name" are non-syntactic names
• E.g. "nsw_yield", "var1" and ".valid" are syntactic names
• Non-syntactic names must be referred with a backtick

```r
tmp <- tibble(`=non-syntatic` = 1:2)
tmp$`=non-syntatic`
## [1] 1 2
`1` <- 2
`1`
## [1] 2
```

Syntactic names consist of letters, digits, . and only and begin with letters or . and also cannot be in reserved words list (?Reserved)
It's usually easier to transform names to syntactic names rather than constantly referring to them by using backticks.

But rename one at a time is a pain.

The `clean_names` function in `janitor` is super handy.

```r
ns %>%
clean_names()
```

## A tibble: 3 x 3
## `Frog id` `weight (kg)` `1980`
## <int>         <dbl>  <dbl>
## 1         1            40    4.3
## 2         2            23    3
## 3         3             4    1.5

I use this A LOT.
Adorn tables with janitor
• You can make nicer looking tables for publication using the `adorn_*` functions in `janitor`.

```r
count %>%
adorn_totals(c("row", "col")) %>%
adorn_percentages() %>%
adorn_pct_formatting(digits = 0) %>%
adorn_ns("front")
```

```r
count

## # A tibble: 3 x 3
##   site `1980` `1981`
##   <chr> <dbl> <dbl>
## 1 A     40     30
## 2 B     20     40
## 3 C     10     10
```

<table>
<thead>
<tr>
<th>site</th>
<th>1980</th>
<th>1981</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>40 (57%)</td>
<td>30 (43%)</td>
<td>70 (100%)</td>
</tr>
<tr>
<td>B</td>
<td>20 (33%)</td>
<td>40 (67%)</td>
<td>60 (100%)</td>
</tr>
<tr>
<td>C</td>
<td>10 (50%)</td>
<td>10 (50%)</td>
<td>20 (100%)</td>
</tr>
<tr>
<td>Total</td>
<td>70 (47%)</td>
<td>80 (53%)</td>
<td>150 (100%)</td>
</tr>
</tbody>
</table>

34/36
If you installed the `dwexercise` package, run below in your R console

```r
learnr::run_tutorial("day2-exercise-01", package = "dwexercise")
```

If the above doesn't work for you, go [here](#).

❓ Questions or issues, let us know!
These slides are licensed under

[ Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International ]