Tidy Data

Max Turgeon

DATA 2010–Tools and Techniques in Data Science

Lecture Objectives

- Understand the concept of tidy data, and give examples of tidy vs. messy data.
- Use functions from the `tidyverse` to turn messy data into tidy data.

Motivation

- Most datasets need to be transformed before being analyzed.
- Tidy data is a framework that allows you to:
    - Visualize, summarize, and analyze data effectively.
    - Follow a set of principles and understand *how* you should transform your data.
- The concept is general, but it is a founding principle of the `tidyverse`.

Definition

- Reference: Wickham (2014), “Tidy Data”. *Journal of Statistical Software*.
- **Tidy data** should follow three principles:
    - Each variable forms a column.
    - Each observation forms a row.
    - Each type of observational unit forms a table.
- This is related to Codd’s third normal form (see COMP 3380)

Example i

	treatmenta	treatmentb
John Smith	—	2
Jane Doe	16	11
Mary Johnson	3	1

Table 1: Typical presentation dataset.

	John Smith	Jane Doe	Mary Johnson
treatmenta	—	16	3
treatmentb	2	11	1

Table 2: The same data as in Table 1 but structured differently.

Example ii

person	treatment	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

Table 3: The same data as in Table 1 but with variables in columns and observations in rows.

Why is this important?

- It makes it easier to summarize.
    - E.g. By having `treatment` as a column, we can group by `treatment` and compute the average `result`.
- As we'll see in a few weeks, it also makes it easier to visualize and analyze the data.
- Finally, it is also helpful to have a consistent way of representing data.
    - Easier to write reusable code.

What can go wrong?

- Wickham (2014) highlights five ways in which data can be messy:
    - Column headers are values, not variable names.
    - Multiple variables are stored in one column.
    - Variables are stored in both rows and columns.
    - Multiple types of observational units are stored in the same table.
    - A single observational unit is stored in multiple tables.
- We'll go over each of them and explain how to address them in R.

Tabular data i

- **Tabular data** (or tabulated data) is a very common way of presenting data.
    - E.g. You have two variables (major and year) and each cell represent the number of observations of a given major and given year.
- Although it is useful summary, it is **not** a tidy format.
- **Why?** Because one variable is spread across multiple columns.
- Let's look at an example from `mtcars`. The columns correspond to the number of gears.

Tabular data ii

```
## Number of cars by number of cylinders and gears

##  cyl   3 4 5
##    4   1 8 2
##    6   2 4 1
##    8  12 0 2
```

Tabular data iii

- Now here's the same data but in tidy format.

```
library(tidyverse)

mtcars |>
  count(cyl, gear)
```

```
##   cyl gear  n
## 1   4    3  1
## 2   4    4  8
## 3   4    5  2
## 4   6    3  2
```

Tabular data iv

```
## 5      6        4   4
## 6      6        5   1
## 7      8        3  12
## 8      8        5   2
```

Pivoting

- How do we turn tabular data into tidy data? By pivoting our dataset.
- There are two types of pivoting:
    - **Pivoting long**: Pivot the column names to values.
    - **Pivoting wide**: Pivot values into column names.
- In other words:
    - Pivot long to go from tabular to tidy.
    - Pivot wide to go from tidy to tabular.

Example i

```r
library(tidyverse)
# Create test dataset
dataset <- tribble(
  ~cyl, ~"3", ~"4", ~"5",
  4, 1, 8, 2,
  6, 2, 4, 1,
  8, 12, 0, 2
)
```

```r
dataset
```

Example ii

```
## # A tibble: 3 x 4
##     cyl   `3`   `4`   `5`
##   <dbl> <dbl> <dbl> <dbl>
## 1     4     1     8     2
## 2     6     2     4     1
## 3     8    12     0     2
```

```
# Pivot long-from tabular to tidy
dataset |>
  pivot_longer(cols = c(`3`, `4`, `5`),
               names_to = "gear",
               values_to = "count")
```

Example iii

```
## # A tibble: 9 x 3
##     cyl gear  count
##   <dbl> <chr> <dbl>
## 1     4 3         1
## 2     4 4         8
## 3     4 5         2
## 4     6 3         2
## 5     6 4         4
## 6     6 5         1
## 7     8 3        12
## 8     8 4         0
## 9     8 5         2
```

Example of pivot wide i

- How could we create the tabular data in the first place?
  - First, count the number of observations.
  - Then, pivot wide.

```
library(tidyverse)

mtcars |>
  group_by(cyl, gear) |>
  summarise(count = n()) |>
  pivot_wider(names_from = "gear",
              values_from = "count")
```

Example of pivot wide ii

```
## # A tibble: 3 x 4
## # Groups:   cyl [3]
##     cyl   `3`   `4`   `5`
##   <dbl> <int> <int> <int>
## 1     4     1     8     2
## 2     6     2     4     1
## 3     8    12    NA     2
```

- As you can see, there was a missing value: there is no car with 8 cylinders and 4 gears. But we know this should simply be zero.

Example of pivot wide iii

```
# Group by cyl, gear
# Use summarise to count
# Then pivot wide

mtcars |>
  group_by(cyl, gear) |>
  summarise(count = n()) |>
  pivot_wider(names_from = "gear",
              values_from = "count",
              values_fill = 0) # Fill missing with 0
```

Example of pivot wide iv

```
## # A tibble: 3 x 4
## # Groups:   cyl [3]
##     cyl   `3`   `4`   `5`
##   <dbl> <int> <int> <int>
## 1     4     1     8     2
## 2     6     2     4     1
## 3     8    12     0     2
```

Exercise

Tidy the simple data frame below. Do you need to make it wider or longer? What variables are you pivoting?

```
preg <- tribble(
  ~pregnant, ~male, ~female,
  "yes",     NA,    10,
  "no",      20,    12
)
```

Solution i

- You need to make it *longer*, so that the values `male` and `female` can be stored in cells.

```
preg |>
  pivot_longer(cols = c("male", "female"),
               names_to = "sex",
               values_to = "count")
```

Solution ii

```
## # A tibble: 4 x 3
##   pregnant sex    count
##   <chr>    <chr>  <dbl>
## 1 yes      male      NA
## 2 yes      female    10
## 3 no       male      20
## 4 no       female    12
```

Multiple variables in one column

- Some columns may in fact include two (or more) variables.
  - E.g. In tabular data, socio-demographic subgroups may contain sex and age information
  - `M20-29`, `F20-39`, etc.
- Solution: Separate the values into different columns
- Note: Often these values are column names, and so a first step may be to pivot the data long.

Example i

- To do this, the `tidyverse` provides the very convenient function `separate()`.
- Let's look at an example, where `rate` contains both cases and population information.

Example ii

```r
library(tidyverse)
dataset <- tribble(
  ~country, ~year, ~rate,
  "Afghanistan", 1999,  "745/19987071",
  "Afghanistan", 2000,  "2666/20595360",
  "Brazil",      1999,  "37737/172006362",
  "Brazil",      2000,  "80488/174504898",
  "China",       1999,  "212258/1272915272",
  "China",       2000,  "213766/1280428583"
)
```

Example iii

```r
dataset |>
  separate(rate, into = c("cases", "population"))
```

```
## # A tibble: 6 x 4
##   country      year cases  population
##   <chr>       <dbl> <chr>  <chr>
## 1 Afghanistan  1999 745    19987071
## 2 Afghanistan  2000 2666   20595360
## 3 Brazil       1999 37737  172006362
## 4 Brazil       2000 80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Example iv

- By default, splitting occurs at any non-alphanumeric character
    - E.g. / , . - |
- You can also specify explicitly at which character the splitting should occur.
- You can also specify whether the splitted values should be converted (when possible).

```
dataset |>
  separate(rate, into = c("cases", "population"),
           sep = "/", convert = TRUE)
```

Example v

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <dbl>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Exercise

Pivot the following dataset long, and then separate sex and age information from the column names.

Hint: Look at the help page for `separate()`, under the `sep` argument. What happens if `sep` is a numeric?

```
dataset <- tribble(
  ~Country, ~m014, ~m1524, ~f014, ~f1524,
  "USA", 2, 4, 4, 6,
  "France", 52, 228, 183, 149
)
```

Solution i

```r
# Socio-demographic variable -> socio
# Cell values -> value (don't know what they represent)
dataset |>
  pivot_longer(cols = c("m014", "m1524",
                        "f014", "f1524"),
               names_to = "socio",
               values_to = "value") |>
  separate(socio, into = c("sex", "age"),
           sep = 1)
```

Solution ii

```
## # A tibble: 8 x 4
##   Country sex   age   value
##   <chr>   <chr> <chr> <dbl>
## 1 USA     m     014       2
## 2 USA     m     1524      4
## 3 USA     f     014       4
## 4 USA     f     1524      6
## 5 France  m     014      52
## 6 France  m     1524    228
## 7 France  f     014     183
## 8 France  f     1524    149
```

Variables are stored in rows and columns i

- This is probably the most complicated type of untidy data.
- Because variables are stored in columns, we need to pivot long.
- But if variable names also appear in cells, we'll need to pivot wide

Variables are stored in rows and columns ii

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
MX17004	2010	1	tmax	—	—	—	—	—	—	—	—
MX17004	2010	1	tmin	—	—	—	—	—	—	—	—
MX17004	2010	2	tmax	—	27.3	24.1	—	—	—	—	—
MX17004	2010	2	tmin	—	14.4	14.4	—	—	—	—	—
MX17004	2010	3	tmax	—	—	—	—	32.1	—	—	—
MX17004	2010	3	tmin	—	—	—	—	14.2	—	—	—
MX17004	2010	4	tmax	—	—	—	—	—	—	—	—
MX17004	2010	4	tmin	—	—	—	—	—	—	—	—
MX17004	2010	5	tmax	—	—	—	—	—	—	—	—
MX17004	2010	5	tmin	—	—	—	—	—	—	—	—

id	date	element	value
MX17004	2010-01-30	tmax	27.8
MX17004	2010-01-30	tmin	14.5
MX17004	2010-02-02	tmax	27.3
MX17004	2010-02-02	tmin	14.4
MX17004	2010-02-03	tmax	24.1
MX17004	2010-02-03	tmin	14.4
MX17004	2010-02-11	tmax	29.7
MX17004	2010-02-11	tmin	13.4
MX17004	2010-02-23	tmax	29.9
MX17004	2010-02-23	tmin	10.7

id	date	tmax	tmin
MX17004	2010-01-30	27.8	14.5
MX17004	2010-02-02	27.3	14.4
MX17004	2010-02-03	24.1	14.4
MX17004	2010-02-11	29.7	13.4
MX17004	2010-02-23	29.9	10.7
MX17004	2010-03-05	32.1	14.2
MX17004	2010-03-10	34.5	16.8
MX17004	2010-03-16	31.1	17.6
MX17004	2010-04-27	36.3	16.7
MX17004	2010-05-27	33.2	18.2

Example i

```r
dataset <- tribble(
  ~Country, ~Stat, ~Y2020, ~ Y2021,
  "Nigeria", "GDP", 400, 450,
  "Nigeria", "Debt", 1200, 1300,
  "Bangladesh", "GDP", 350, 400,
  "Bangladesh", "Debt", 100, 150
)

dataset
```

Example ii

```
## # A tibble: 4 x 4
##   Country    Stat  Y2020 Y2021
##   <chr>      <chr> <dbl> <dbl>
## 1 Nigeria    GDP     400   450
## 2 Nigeria    Debt   1200  1300
## 3 Bangladesh GDP     350   400
## 4 Bangladesh Debt    100   150
```

```
dataset |>
  pivot_longer(cols = c("Y2020", "Y2021"),
               names_to = "year", values_to = "value")
  pivot_wider(names_from = "Stat", values_from = "value
```

Example iii

```
## # A tibble: 4 x 4
##   Country    year    GDP  Debt
##   <chr>      <chr> <dbl> <dbl>
## 1 Nigeria    Y2020   400  1200
## 2 Nigeria    Y2021   450  1300
## 3 Bangladesh Y2020   350   100
## 4 Bangladesh Y2021   400   150
```

Multiple types in same dataset i

- The example below contains values in the column names

year	artist	track	time	date.entered	wk1	wk2	wk3
2000	2 Pac	Baby Don't Cry	4:22	2000-02-26	87	82	72
2000	2Ge+her	The Hardest Part Of ...	3:15	2000-09-02	91	87	92
2000	3 Doors Down	Kryptonite	3:53	2000-04-08	81	70	68
2000	98^0	Give Me Just One Nig...	3:24	2000-08-19	51	39	34
2000	A*Teens	Dancing Queen	3:44	2000-07-08	97	97	96
2000	Aaliyah	I Don't Wanna	4:15	2000-01-29	84	62	51
2000	Aaliyah	Try Again	4:03	2000-03-18	59	53	38
2000	Adams, Yolanda	Open My Heart	5:30	2000-08-26	76	76	74

- But if you tidy it, then information about each track is repeated.
    - Year, artist, track, time, date.entered.

Multiple types in same dataset ii

- A better strategy is to keep the track information and Billboard information *in separate datasets*.
    - Leads to easier maintenance.
- Once we're ready to analyze, we can join them back (more on this next week).

Multiple types in same dataset iii

id	artist	track	time
1	2 Pac	Baby Don't Cry	4:22
2	2Ge+her	The Hardest Part Of ...	3:15
3	3 Doors Down	Kryptonite	3:53
4	3 Doors Down	Loser	4:24
5	504 Boyz	Wobble Wobble	3:35
6	98^0	Give Me Just One Nig...	3:24
7	A*Teens	Dancing Queen	3:44
8	Aaliyah	I Don't Wanna	4:15
9	Aaliyah	Try Again	4:03
10	Adams, Yolanda	Open My Heart	5:30
11	Adkins, Trace	More	3:05
12	Aguilera, Christina	Come On Over Baby	3:38
13	Aguilera, Christina	I Turn To You	4:00
14	Aguilera, Christina	What A Girl Wants	3:18
15	Alice Deejay	Better Off Alone	6:50

id	date	rank
1	2000-02-26	87
1	2000-03-04	82
1	2000-03-11	72
1	2000-03-18	77
1	2000-03-25	87
1	2000-04-01	94
1	2000-04-08	99
2	2000-09-02	91
2	2000-09-09	87
2	2000-09-16	92
3	2000-04-08	81
3	2000-04-15	70
3	2000-04-22	68
3	2000-04-29	67
3	2000-05-06	66

One type in multiple tables

- The opposite of the previous setup.
- For example, you receive an Excel file, where each spreadsheet has the same data but for different years.
- **Solution**: Load each spreadsheet, add a variable (e.g. year) and “stack” the data frames.

Example i

```
# First year of data
data_year1 <- tribble(
  ~Country, ~sex, ~value,
  "USA", "male", 2,
  "USA", "female", 4,
  "France", "male", 52,
  "France", "female", 183
)
```

Example ii

```r
# Second year of data
# Note: column names are different!
data_year2 <- tribble(
    ~Country, ~sex, ~score,
  "USA", "male", 4,
  "USA", "female", 6,
  "France", "male", 228,
  "France", "female", 149
)
```

Example iii

```
# First change name, then
# create new variable
data_year1 <- data_year1 |>
  rename(score = value) |> # old = new
  mutate(year = 1)

data_year2 <- data_year2 |>
  mutate(year = 2)
```

```
# Then stack them
bind_rows(data_year1, data_year2)
```

Example iv

```
## # A tibble: 8 x 4
##   Country sex    score  year
##   <chr>   <chr>  <dbl> <dbl>
## 1 USA     male       2     1
## 2 USA     female     4     1
## 3 France  male      52     1
## 4 France  female   183     1
## 5 USA     male       4     2
## 6 USA     female     6     2
## 7 France  male     228     2
## 8 France  female   149     2
```

Summary

- We learned about tidy data:
    - Each variable has its own column
    - Each observation has its own row
    - Each value has its own cell
- Tidy data is about efficient data analysis
    - It's not necessarily the most memory efficient way of storing data…
- Not all `R` functions are meant to be used with tidy data
    - But the tidyverse (and `tidymodels`) is optimized for it.