DSA2101

Essential Data Analytics Tools: Data Visualization

Yuting Huang

Week 6 Tidy data

Midterm: Tuesday March 7, 8-9:30am at LT32

Things to bring on the exam day:

- A laptop with the latest `R`, `RStudio`, and Examplify installed.
- The laptop charger.
- Your NUS matriculation card.
- **Arrive at least 10 minutes early** on the exam day.

Common briefing sessions (if you are new to Examplify):

https://wiki.nus.edu.sg/display/DA/Common+Briefing+Sessions

Announcements (Midterm test)

Format of the exam: **open-book, block-internet**.

- You can use your notes in the exam, but will **not** have access to the internet.

The questions will be similar to what you've seen in tutorials and in-class exercises.

- There will be some sample questions at the end of the next tutorial worksheet to help guide your preparation for the midterm.
- Solve them by yourself first.
- We will go through those questions during lecture in Week 7.

Requirements on your `Rmd` file

Answer all questions in a single `R` Markdown file.

- Make sure you follow all the instructions.
- Remember to use the usual relative path setting.
- Ensure that your `Rmd` file can knit to HTML before you submit anything to Examplify or Canvas.

Requirements on your `Rmd` file

We would deduct marks if we are not able to run/knit your `Rmd` file. Reasons for marks deduction include:

- Data set not read in.
- syntax errors in the code.
- Required objects not found within the code.
- Explanations/discussions not written under `Rmd` text sections of the required title(s).

Road map

1. **Data transformation** Week 5
   - `filter()`, `select()`, `mutate()`, `arrange()`, and `summarize()`
   - `group_by()` and `%>%`
2. **Tidy data** Week 6
   - `gather()`, `spread()`, `separate()`, and `unite()`
3. **Relational data** Week 7

Tidy data

Artwork by Allison Horst

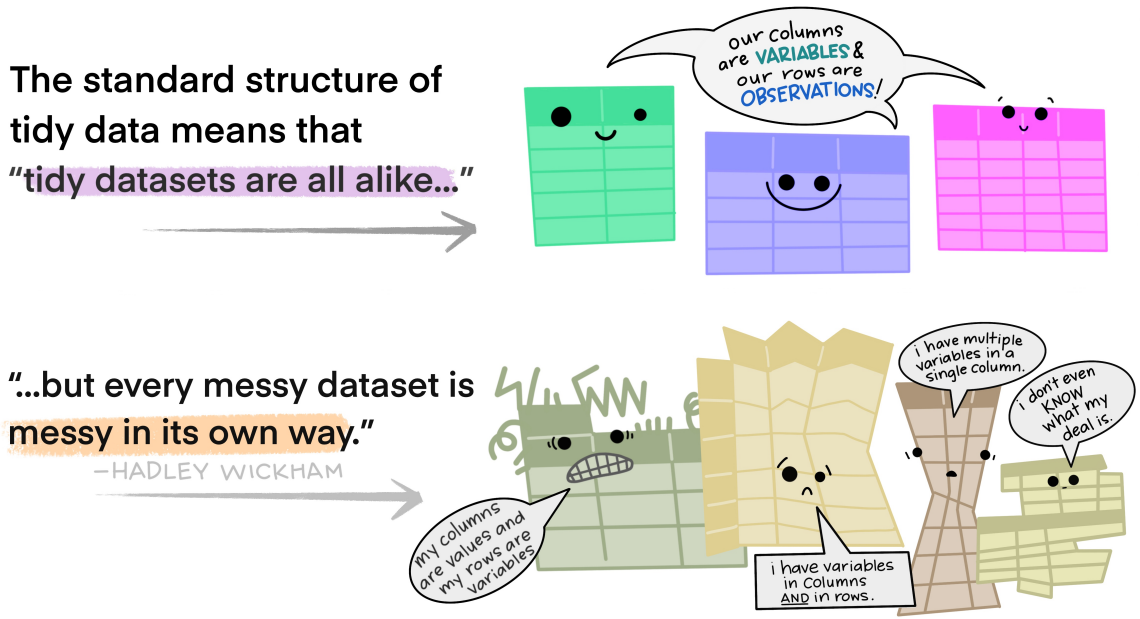
Tidy data

You have definitely heard of the word "tidy" in your life.

- In terms of data analysis, **tidy data** refers to a standard way of mapping the meaning of a data set to its structure
- Getting our data into this format requires some work in the beginning, but the payoff is in the ease with which we will be able to manipulate it afterwards.

Data structure

Most statistical data sets are *rectangular tables* made up of rows and columns.



Artwork by Allison Horst

Same data, two different structures

The same data set can be presented in many different ways.

Name	**Treatment A**	**Treatment B**
John Smith	-	2
Jane Doe	16	11
Mary Johnson	3	1

Treatment	**John Smith**	**Jane Doe**	**Mary Johnson**
A	-	16	3
B	2	11	1

Data semantics

A data set is a collection of values.

1. Values are organized in two ways: every value belongs to a variable (column) and an observation (row).
2. A variable contains all values that measure the same underlying attribute across observations.
3. An observation contains all values measured across attributes.

Data semantics

In the previous example, there are three variables:

- person's name, with three possible values
- treatment, with two possible values
- result, with six possible values, one of which is missing

There are six observation units. Each one is identified by the combination of person and treatment values.

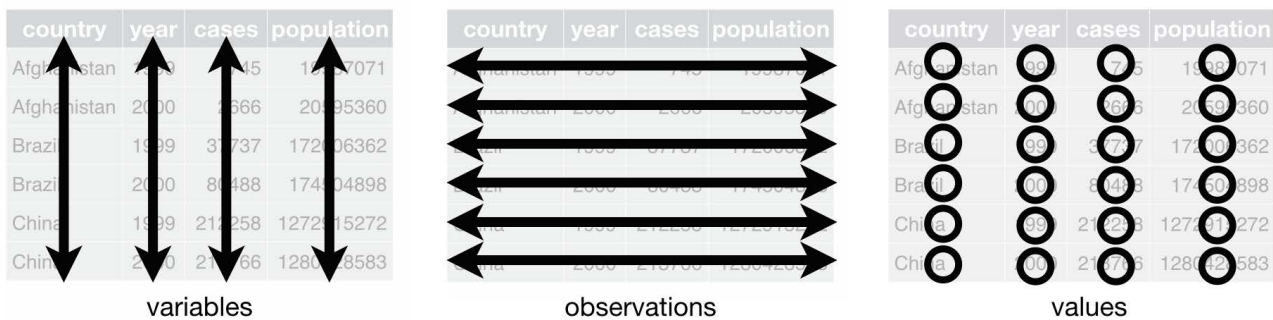
Tidy version of the data set

id	Name	Treatment	Result
1	John Smith	A	-
2	Jane Doe	A	16
3	Mary Johnson	A	3
4	John Smith	B	2
5	Jane Doe	B	11
6	Mary Johnson	B	1

Definition of tidy data

Tidy data is a standard way of structuring a data. It requires that

- Each variable forms a column.
- Each observation forms a row.
- Each type of observational unit forms a table.



Another example

Examine the following table on Stock prices. Are these tidy data?

Date	**Amazon**	**Boeing**	**Google**
2009-01-01	174.90	173.55	174.34
2009-01-02	171.42	172.61	170.04

Although the data are organized in a rectangular spreadsheet, they do not follow the definition for "tidy" format.

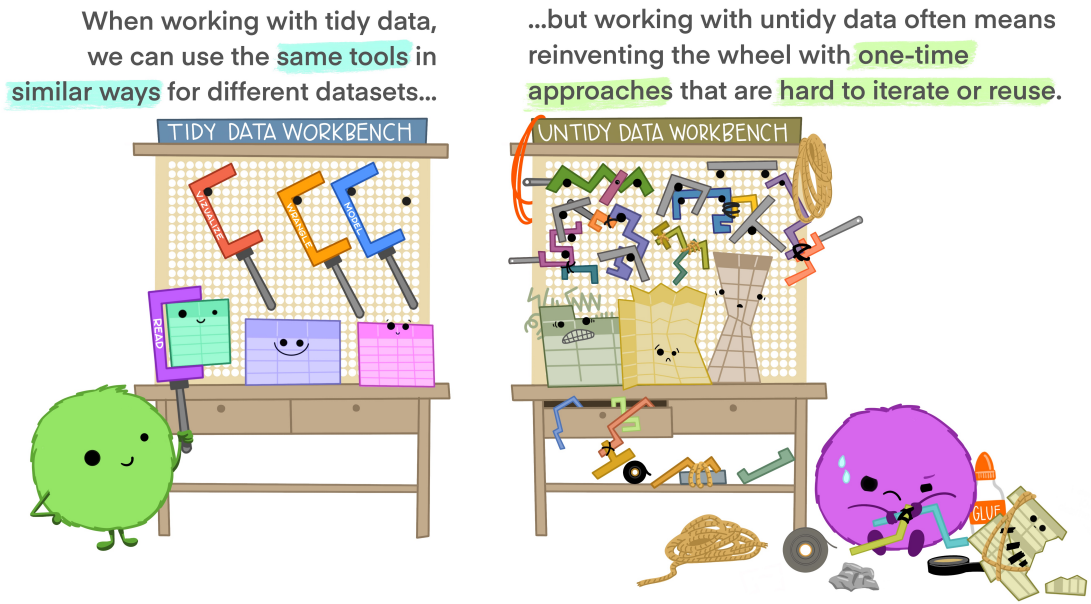
Tidy version of the data set

In a tidy data set, each variable should have its own column, and each observation should form its own row.

Date	**Stock**	**Prices**
2009-01-01	Amazon	174.90
2009-01-01	Boeing	173.55
2009-01-01	Google	174.30
2009-01-02	Amazon	171.42
2009-01-02	Boeing	172.61
2009-01-02	Google	170.04

Why tidy data?

- Placing variables in column allows the vectorized nature of R functions to take precedence.
- Having a consistent data structure also means that we don't have to re-invent the tools to work with data.



Artwork by Allison Horst

How to order variables?

A good ordering of variables makes it easier to scan the raw values.

- **Fixed variables** refer to those that describe the data set. These are typically known in advance. They should come first.
  - Example: id, name, gender
- **Measured variables** are what we actually measure. These should come later.
  - Example: height, weight, blood sugar level, GDP

Messy data

Data can be untidy in many different ways. These are the two most common ones:

- Column headers are values, instead of variable names.
  - We can solve this problem using `gather()`
- Multiple variables are stored in one column.
  - Solve this using `spread()`

Tuberculosis (TB) Cases example

The following tibble contains TB counts for three countries from the `tidyverse` library.

```
library(tidyverse)
table4a
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
##   <chr>        <dbl>  <dbl>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

TB Cases example

Notice that there are three variables:

- Country, with 3 values
- Year, with 2 values
- Number of TB cases, with 6 distinct values.

Problems to fix:

1. One of the variables, Year, is stored in the column names. While this is okay for display, it is not tidy.
2. The number of TB cases is spread across columns too!

The `gather()` function

We need to `gather` the columns into a new pair of variables. The information that the function requires are

1. The set of columns that represent values, not variables
   - The columns `1999` and `2000`
2. The name of the variable that will be created whose values form the column names for now.
   - This is the `key` argument to `gather()`. In this example we should call this variable **year**.
3. The name of the variable that will be created whose values are currently residing in the cells of the data set.
   - This is the `value` argument to `gather()`. In this example we should call this variable **cases**.

How to gather

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

The gather() function

```
table4a %>%
  gather(`1999`:`2000`, key = "year", value = "cases")
```

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

The `spread()` function

The following tibble contains an observation unit scattered across rows:

```
head(table2, n = 3)
```

```
## # A tibble: 3 x 4
##   country      year type           count
##   <chr>       <dbl> <chr>          <dbl>
## 1 Afghanistan  1999 cases            745
## 2 Afghanistan  1999 population 19987071
## 3 Afghanistan  2000 cases           2666
```

- The observation unit should be a country in each year
- The `type` column contains two different measurements for each ideal observation unit.

The `spread()` function

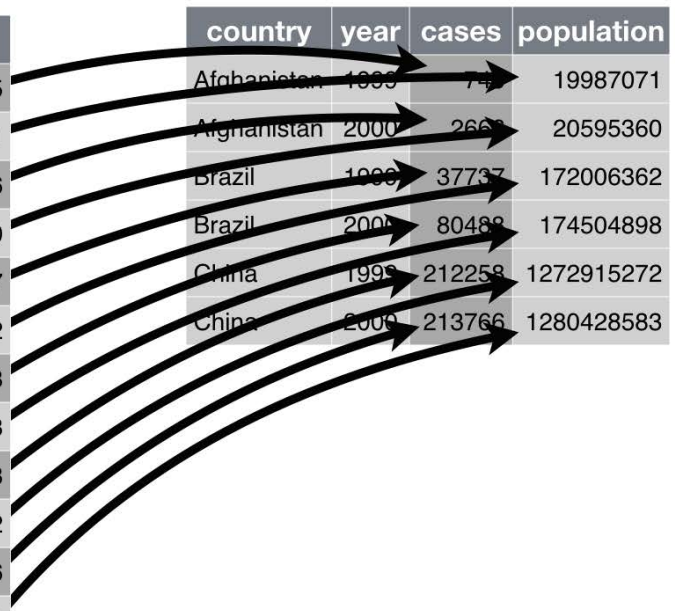
We need to spread the `type` and `count` columns out into a single row for each observation unit.

1. The column that currently contains variable names is `type`.
   - This is the `key` argument to the `spread()` function.
2. The column that currently contains from multiple variables is `count`.
   - This is passed as the `value` argument.

How to spread

country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

table2



The `spread()` function

```
table2 %>%
  spread(key = type, value = count)
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 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
```

`spread()` vs. `gather()`

The two are complementary operations.

- `gather()` makes a data set tall and narrow.
- `spread()` makes a data set wide and short.

Artwork by Allison Horst

The `pivot` functions

There are two functions that are currently in development:

- `pivot_longer()`
- `pivot_wider()`.

We shall also introduce them as they are easy to use and contains slightly more features.

The `pivot` functions

Recall the problem with `table4a`:

- Column names are not names of variables, but *values* of a variable.

To tidy a data set like this, we can also use `pivot_longer()`.

```
table4a %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "cases")
```

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

The `pivot` functions

Recall the problem with `table2`:

- The unit of observation should be a country in a year. But each observation is spread across two rows.

To tidy a data set like this, we can also use `pivot_wider()`.

```
table2 %>%
  pivot_wider(names_from = type, values_from = count)
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 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
```

The `pivot` functions

- `pivot_longer()` makes the data set *longer* by increasing the number of rows and decreasing the number of columns.
- `pivot_wider()` makes the data set *wider* by increasing the number of columns and decreasing the number of rows.
- `pivot_wider()` is the inverse transformation of `pivot_longer()`.

`separate()` a column

The same TB data could also be stored like this.

```
table3
```

```
## # A tibble: 6 x 3
##   country      year rate
##   <chr>       <dbl> <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
```

- Instead of TB counts and population, they are combined as a `rate` variable.
- Need to tease them apart with `separate()`

How to `separate`

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

table3

country	year	cases	population
Afghanistan	1999	**745**	19987071
Afghanistan	2000	**2666**	20595360
Brazil	1999	**37737**	172006362
Brazil	2000	**80488**	174504898
China	1999	**212258**	1272915272
China	2000	**213766**	1280428583

The `separate()` function

`separate()` pulls apart one column into multiple columns, by splitting wherever a separator character appears.

```
table3 %>%
  separate(rate, into = c("cases", "population"), convert = TRUE)
```

```
## # 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
```

- `convert = TRUE` converts the new columns to the appropriate type.

The `unite()` function

The opposite of separating columns is uniting them.

- `unite()` combines multiple columns into one, using a separator character.
- You will need it much less frequently than `separate()`, but it is still a usefull tool to have in your back pocket.

The `unite()` function

- For instance, the `year` variable is split into `century` and `year`.
- We can use `unite()` to rejoin these columns.

```
table5
```

```
## # A tibble: 6 x 4
##   country     century year  rate
##   <chr>       <chr>   <chr> <chr>
## 1 Afghanistan 19      99    745/19987071
## 2 Afghanistan 20      00    2666/20595360
## 3 Brazil      19      99    37737/172006362
## 4 Brazil      20      00    80488/174504898
## 5 China       19      99    212258/1272915272
## 6 China       20      00    213766/1280428583
```

How to `unite`

country	year	rate
Afghanistan	19**99**	745 / 19987071
Afghanistan	20**00**	2666 / 20595360
Brazil	19**99**	37737 / 172006362
Brazil	20**00**	80488 / 174504898
China	19**99**	212258 / 1272915272
China	20**00**	213766 / 1280428583

country	century	year	rate
Afghanistan	19	**99**	745 / 19987071
Afghanistan	20	**0**	2666 / 20595360
Brazil	19	**99**	37737 / 172006362
Brazil	20	**0**	80488 / 174504898
China	19	**99**	212258 / 1272915272
China	20	**0**	213766 / 1280428583

The `unite()` function

```
table5 %>%
  unite(col = year, century, year, sep = "")
```

```
## # A tibble: 6 x 3
##   country     year  rate
##   <chr>       <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
```

- `sep = ""` specifies the separator to use between values. In this example, we unite the values without any space.

Summary on tidy data

The principles of tidy data may seem very obvious now. You might wonder if you will ever encounter a data set that is not tidy.

Unfortunately, however, most data that you will encounter will be untidy. There are two main reasons.

- Most people are not familiar with the principles of tidy data, and it is hard to derive them yourself unless you spent *a lot* of time working with data.
- Data are often organized to facilitate some use, other than analysis. For example, they can be organized to make data entry as easy as possible.

This means that for most real analyses, you will need to do so data tidying.

Summary on tidy data

The first step of tidying data is always to figure out what the variables and observations are.

- Sometimes this is easy.
- Other times you will need to consult the people who originally generated the data.

The second step is to resolve one of two common problems:

- One variable spreads across multiple columns.
- One observation scatters across multiple rows.

Usually a data set will only suffer from one of these problems.

Second summary

`gather()`

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

`spread()`

country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

table2

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

`separate()`

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

table3

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

`unite()`

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

country	century	year	rate
Afghanistan	19	99	745 / 19987071
Afghanistan	20	0	2666 / 20595360
Brazil	19	99	37737 / 172006362
Brazil	20	0	80488 / 174504898
China	19	99	212258 / 1272915272
China	20	0	213766 / 1280428583

Artwork by Allison Horst