Tidy Data

Brad Stieber

2018-10-01

Introduction

Goals for this talk

My goal is for you to walk away from this presentation with an understanding of:

- Tidy data philosophy

Goals for this talk

My goal is for you to walk away from this presentation with an understanding of:

- Tidy data philosophy
- `tidyverse` data terminology

Goals for this talk

My goal is for you to walk away from this presentation with an understanding of:

- Tidy data philosophy
- `tidyverse` data terminology
- Common types of *untidy* data

Goals for this talk

My goal is for you to walk away from this presentation with an understanding of:

- Tidy data philosophy
- `tidyverse` data terminology
- Common types of *untidy* data
- Operations for tidying up

Goals for this talk

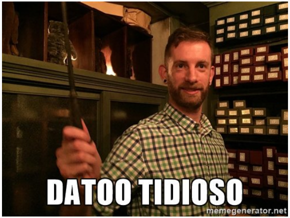
My goal is for you to walk away from this presentation with an understanding of:

- Tidy data philosophy
- `tidyverse` data terminology
- Common types of *untidy* data
- Operations for tidying up
- Displaying tidy data

What this is based off of

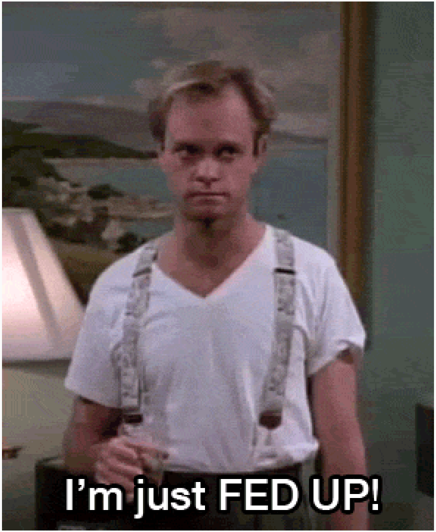
Most of what follows is based off of Hadley Wickham's paper on tidy data. I would strongly recommend reading that paper.

If you're looking for a practical introduction, Hadley Wickham has one of those too.



I also borrow a bit from other resources (which will be listed at the end), as well as my own experience working with tidy *and* untidy datasets.

But mostly. . .



Idea (the theory)

Why tidy data?

- Consistency

Why tidy data?

- Consistency
- Rely on vectorization (in `R` and `pandas`), and expected/desired behavior in grouped aggregation (excel, tableau)

Why tidy data?

- Consistency
- Rely on vectorization (in `R` and `pandas`), and expected/desired behavior in grouped aggregation (excel, tableau)
- Foresight

What is tidy data

There are three qualities a dataset must have to be considered "tidy"

1. Each variable forms a column.

What is tidy data

There are three qualities a dataset must have to be considered “tidy”

1. Each variable forms a column.
2. Each observation forms a row.

What is tidy data

There are three qualities a dataset must have to be considered "tidy"

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

The Language of Tidy Data

- Dataset: a collection of values (e.g. iris data)
- Variable:
- Values:
- Observation:
- Long:
- Wide:

It's usually easy to figure out things like *observations* and *variables* for a given dataset, but defining them in the abstract can be difficult.

The Data Tidying Operations

Getting data into a tidy format first requires understanding the three qualities of tidy data, as well as the five most common types of untidy data.

Then, we can get most forms of untidy data to be tidy by utilizing four "verbs" of data tidying

- `gather`: takes multiple columns, and gathers them into key-value pairs: it makes "wide" data longer

The Data Tidying Operations

Getting data into a tidy format first requires understanding the three qualities of tidy data, as well as the five most common types of untidy data.

Then, we can get most forms of untidy data to be tidy by utilizing four "verbs" of data tidying

- `gather`: takes multiple columns, and gathers them into key-value pairs: it makes "wide" data longer
- `spread`: takes two columns (key & value) and spreads in to multiple columns, it makes "long" data wider

The Data Tidying Operations

Getting data into a tidy format first requires understanding the three qualities of tidy data, as well as the five most common types of untidy data.

Then, we can get most forms of untidy data to be tidy by utilizing four “verbs” of data tidying

- `gather`: takes multiple columns, and gathers them into key-value pairs: it makes “wide” data longer
- `spread`: takes two columns (key & value) and spreads in to multiple columns, it makes “long” data wider
- `separate`: turns a single character column into multiple columns, based on a regular expression or specific positions

The Data Tidying Operations

Getting data into a tidy format first requires understanding the three qualities of tidy data, as well as the five most common types of untidy data.

Then, we can get most forms of untidy data to be tidy by utilizing four "verbs" of data tidying

- `gather`: takes multiple columns, and gathers them into key-value pairs: it makes "wide" data longer
- `spread`: takes two columns (key & value) and spreads in to multiple columns, it makes "long" data wider
- `separate`: turns a single character column into multiple columns, based on a regular expression or specific positions
- `unite`: concatenate multiple columns into one

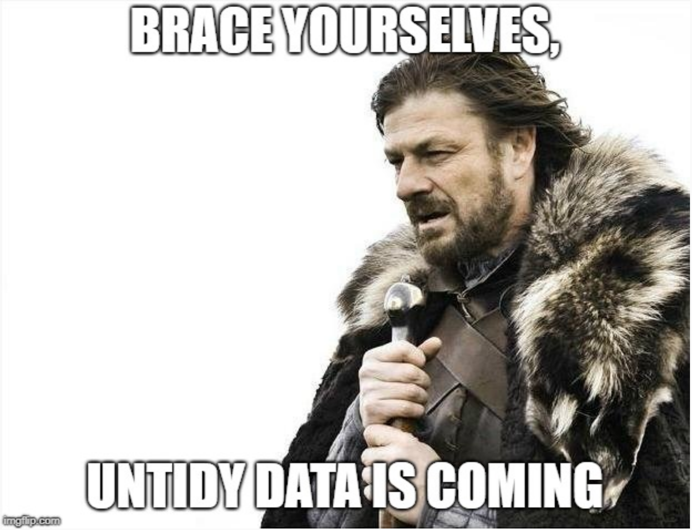
Execution (the practice)

Five common types of untidy data

Here are the five most common types of untidy data you're likely to experience "in the wild".

- 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 through examples of each of the five.


BRACE YOURSELVES,
UNTIDY DATA IS COMING
imgflip.com

1. Column headers are values, not variable names

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

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

How would we tidy up?

Tidying # 1

Need to `gather` columns into key-value (year-cases) pairs:

```
tb_data %>%
    gather(key = year,
           value = tb_cases,
           -country)
```

```
## # A tibble: 6 x 3
##   country     year  tb_cases
##   <chr>       <chr>    <int>
## 1 Afghanistan 1999       745
## 2 Brazil      1999     37737
## 3 China       1999    212258
## # ... with 3 more rows
```

2. Multiple variables are stored in one column

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Hair - Eye - Sex	n
Black - Brown - Male	32
Brown - Brown - Male	53
Red - Brown - Male	10
Blond - Brown - Male	3
Black - Blue - Male	11
Brown - Blue - Male	50

How would we tidy up?

Tidying #2 (1/2)

Need to `separate` one column (`Hair - Eye - Sex`) into multiple columns (`hair`, `eye`, `sex`)

```
hec_untidy %>%
  separate(col = `Hair - Eye - Sex`,
           into = c('hair', 'eye', 'sex'))
```

```
## # A tibble: 6 x 4
##   hair  eye   sex       n
##   <chr> <chr> <chr> <dbl>
## 1 Black Brown Male     32
## 2 Brown Brown Male     53
## 3 Red   Brown Male     10
## # ... with 3 more rows
```

Tidying #2 (2/2)

Could go a step further and use `uncount` (the opposite of `dplyr::count`):

```
hec_untidy %>%
  separate(col = `Hair - Eye - Sex`,
           into = c('hair', 'eye', 'sex')) %>%
  uncount(n)
```

```
## # A tibble: 159 x 3
##   hair  eye   sex
##   <chr> <chr> <chr>
## 1 Black Brown Male
## 2 Black Brown Male
## 3 Black Brown Male
## # ... with 156 more rows
```

3. Variables are stored in both rows and columns

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

year	month	stock	d1	d2	d3	d4
2000	1	CAC	1772.80	1750.50	1718.00	1708.10
2000	1	DAX	1628.75	1613.63	1606.51	1621.04
2000	1	FTSE	2443.60	2460.20	2448.20	2470.40
2000	1	SMI	1678.10	1688.50	1678.60	1684.10

How would we tidy up? Think carefully about that the **observation** is for this data?

Tidying #3 (1/2)

What is the “observation” (a **stock on a day** or a day)?

```r
mkt_untidy %>%
    gather(day, price, starts_with('d')) %>%
    # spread(stock, price) %>%
    mutate(day = gsub('d', '', day)) %>%
    unite(date, year, month, day, sep = '-') %>%
    mutate(date = lubridate::ymd(date))
```

```
## # A tibble: 16 x 3
##   date       stock price
##   <date>     <chr> <dbl>
## 1 2000-01-01 CAC   1773.
## 2 2000-01-01 DAX   1629.
## 3 2000-01-01 FTSE  2444.
## # ... with 13 more rows
```

Tidying #3 (2/2)

What is the "observation" (a stock on a day or **a day**)?

```
mkt_untidy %>%
    gather(day, price, starts_with('d')) %>%
    spread(stock, price) %>%
    mutate(day = gsub('d', '', day)) %>%
    unite(date, year, month, day, sep = '-') %>%
    mutate(date = lubridate::ymd(date))
```

```
## # A tibble: 4 x 5
##   date         CAC   DAX  FTSE   SMI
##   <date>     <dbl> <dbl> <dbl> <dbl>
## 1 2000-01-01 1773. 1629. 2444. 1678.
## 2 2000-01-02 1750. 1614. 2460. 1688.
## 3 2000-01-03 1718  1607. 2448. 1679.
## # ... with 1 more row
```

4. Multiple types of observational units are stored in the same table

This is one that gets violated a lot. Our desire is to have *all* the data in one spot.

Data should be **normalized** during the process of tidying, it is not until we reach the analytical part of our data science process that denormalization should be preferred.

golfer	birth_date	birth_place	tournament_date	tournament	final_score
Tiger Woods	1975-12-30	Cypress, CA	1996-10-06	Las Vegas	-27
Tiger Woods	1975-12-30	Cypress, CA	1996-10-20	Disney	-21
Tiger Woods	1975-12-30	Cypress, CA	1997-01-12	Mercedes	-14
Tiger Woods	1975-12-30	Cypress, CA	1997-04-13	Masters	-18

How would you tidy up?

Tidying #4 (use `dplyr::select`)

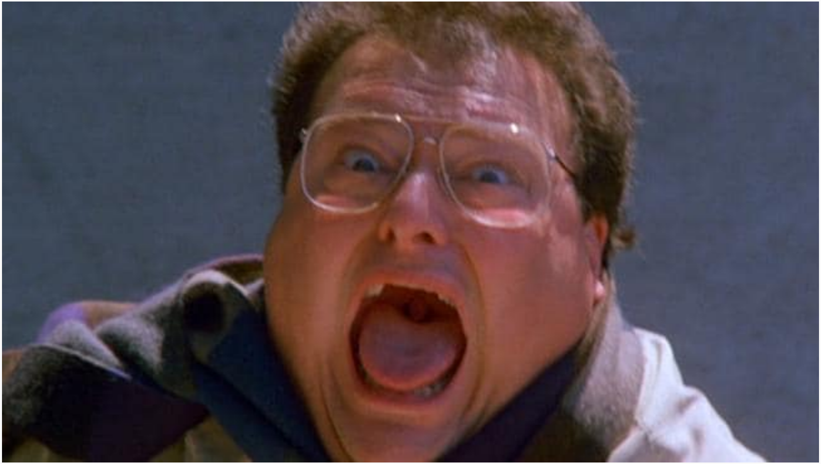
```
# helper function
select_distinct <- function(data, ...){
  select(data, ...) %>%
    distinct()
}
# golfer table
tw_data %>%
  select_distinct(golfer, birth_date, birth_place)

# tournament table
tw_data %>%
  select_distinct(tournament, tournament_date)

# result table
tw_data %>%
  select_distinct(tournament, winner = golfer, final_score)
```

5. A single observational unit is stored in multiple tables

Have you ever worked with US government data before?



5. A single observational unit is stored in multiple tables

Have you ever worked with US government data before? If so, you know this is common:

year	cpi
2015	237

year	cpi
2016	240

year	cpi
2017	245

Not hard to remedy, but still annoying and potentially dangerous. Easy fix for *consistent* tables: `dplyr::bind_rows`

```
bind_rows(t_15, t_16, t_17)
```

```
## # A tibble: 3 x 2
##    year   cpi
##   <dbl> <dbl>
## 1  2015   237
## 2  2016   240
## 3  2017   245
```

Displaying tidy data

OPTIONAL - Organizing data in spreadsheets

Conclusion

4 most important things to remember

1. Put each dataset in a table

4 most important things to remember

1. Put each dataset in a table
    - `data.frame` (`stringsAsFactors`? printing to a console? lazy evaluation?)

4 most important things to remember

1. Put each dataset in a table
   - `data.frame` (`stringsAsFactors`? printing to a console? lazy evaluation?)
   - `tibble` (AKA the nice version of `data.frame`)

4 most important things to remember

1. Put each dataset in a table
    - `data.frame` (`stringsAsFactors`? printing to a console? lazy evaluation?)
    - `tibble` (AKA the nice version of `data.frame`)
2. Put each variable in a column

4 most important things to remember

1. Put each dataset in a table
   - `data.frame` (`stringsAsFactors`? printing to a console? lazy evaluation?)
   - `tibble` (AKA the nice version of `data.frame`)
2. Put each variable in a column
3. Ask yourself the following questions

4 most important things to remember

1. Put each dataset in a table
    - `data.frame` (`stringsAsFactors`? printing to a console? lazy evaluation?)
    - `tibble` (AKA the nice version of `data.frame`)
2. Put each variable in a column
3. Ask yourself the following questions
    - What are the rows of my dataset (level of detail)?

4 most important things to remember

1. Put each dataset in a table
   - `data.frame` (`stringsAsFactors`? printing to a console? lazy evaluation?)
   - `tibble` (AKA the nice version of `data.frame`)
2. Put each variable in a column
3. Ask yourself the following questions
   - What are the rows of my dataset (level of detail)?
   - Is each column a *distinct* variable?

4 most important things to remember

1. Put each dataset in a table
    - `data.frame` (`stringsAsFactors`? printing to a console? lazy evaluation?)
    - `tibble` (AKA the nice version of `data.frame`)
2. Put each variable in a column
3. Ask yourself the following questions
    - What are the rows of my dataset (level of detail)?
    - Is each column a *distinct* variable?
    - How hard would it be to calculate a grouped aggregation?

4 most important things to remember

1. Put each dataset in a table
   - `data.frame` (`stringsAsFactors`? printing to a console? lazy evaluation?)
   - `tibble` (AKA the nice version of `data.frame`)
2. Put each variable in a column
3. Ask yourself the following questions
   - What are the rows of my dataset (level of detail)?
   - Is each column a *distinct* variable?
   - How hard would it be to calculate a grouped aggregation?
4. Structure and tidy up your data to be manipulated by a computer. Ignore urges to make it easily viewed by a human.

Wrapping up

If I had one thing to tell biologists learning bioinformatics, it would be "write code for humans, write data for computers". - Vince Buffalo

Other resources