Tidy Data

Brad Stieber

Prepared for 2020-04-01 Meeting with New Analytics Team Members

You can find the repository for this presentation on my GitHub.

The only package needed:

```r
library(tidyverse)
```

Introduction

Goals for this talk

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

- Tidy data philosophy

Goals for this talk

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

- Tidy data philosophy
- Some `tidyverse` data terminology

Goals for this talk

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

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

Goals for this talk

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

- Tidy data philosophy
- Some `tidyverse` data terminology
- Common types of *untidy* data
- Operations for tidying data in `R`

Goals for this talk

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

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

Where did this come from?

Most of what follows is based off of Hadley Wickham's paper on tidy data.

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

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

Idea (the theory)

- 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

Why tidy data?

- Consistency
- Rely on vectorization (in R and pandas), and expected/desired behavior in grouped aggregation (excel, tableau)
- Foresight
- It's never been easier

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: all values that measure the same underlying attribute (e.g. height, width)
- Values: a specific measurement or attribute for a variable (e.g. $100)
- Observation: all values measured on the same unit (like a person, or a day, or a game) across variables

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 requires understanding the three qualities of tidy data, as well as the five most common types of untidy data (more on these shortly).

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 requires understanding the three qualities of tidy data, as well as the five most common types of untidy data (more on these shortly).

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
    - See also: pivot_longer (supersedes gather as of September 2019)

The Data Tidying Operations

Getting data into a tidy format requires understanding the three qualities of tidy data, as well as the five most common types of untidy data (more on these shortly).

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
    - See also: `pivot_longer` (supersedes `gather` as of September 2019)
- `spread`: takes two columns (key & value) and spreads into multiple columns - it makes "long" data wider

The Data Tidying Operations

Getting data into a tidy format requires understanding the three qualities of tidy data, as well as the five most common types of untidy data (more on these shortly).

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
  - See also: `pivot_longer` (supersedes gather as of September 2019)
- `spread`: takes two columns (key & value) and spreads into multiple columns - it makes "long" data wider
  - See also: `pivot_wider` (supersedes spread as of September 2019)

The Data Tidying Operations

Getting data into a tidy format requires understanding the three qualities of tidy data, as well as the five most common types of untidy data (more on these shortly).

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
    - See also: `pivot_longer` (supersedes gather as of September 2019)
- `spread`: takes two columns (key & value) and spreads into multiple columns - it makes "long" data wider
    - See also: `pivot_wider` (supersedes spread as of September 2019)
- `separate`: turns a single column that is character-valued into multiple columns, based on a regular expression or specific positions

The Data Tidying Operations

Getting data into a tidy format requires understanding the three qualities of tidy data, as well as the five most common types of untidy data (more on these shortly).

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
  - See also: `pivot_longer` (supersedes gather as of September 2019)
- `spread`: takes two columns (key & value) and spreads into multiple columns - it makes "long" data wider
  - See also: `pivot_wider` (supersedes spread as of September 2019)
- `separate`: turns a single column that is character-valued 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.

For each untidy dataset...

Think about the following questions:

- Why is it untidy?
- How would you work with it without tidying it first (e.g. lots of calculated flags, hard coding, black magic)?
- What are the variables, values, and observations?
- How can we tidy it up?
- How would you work with it in its tidy form?

1. Column headers are values, not variable names

The first dataset we'll look at comes from the WHO and displays the number of TB cases for three countries in two years.

country	1999	2000
Afghanistan	745	2,666
Brazil	37,737	80,488
China	212,258	213,766

This data is too *wide*, as 1999 and 2000 are **values** for a **variable** we could call year.

Although difficult to analyze, this format is helpful for presentation and data entry.

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
```

Tidying #1 (using `pivot_longer`)

```
tb_data %>%
    pivot_longer(cols = -country,
                 names_to = "year",
                 values_to = "tb_cases")
```

```
## # A tibble: 6 x 3
##   country     year  tb_cases
##   <chr>       <chr>    <int>
## 1 Afghanistan 1999       745
## 2 Afghanistan 2000      2666
## 3 Brazil      1999     37737
## # ... with 3 more rows
```

2. Multiple variables are stored in one column

The next table has two columns, but it should have four. How would you work with this data without tidying it first?

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

The Hair - Eye - Sex **variable** actually has **values** for three separate **variables** stored within it.

Tidying #2 (1/2)

We 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'),
           sep = ' - ') # actually not needed
```

```
## # 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`):

```r
hec_untidy %>%
  separate(col = `Hair - Eye - Sex`,
           into = c('hair', 'eye', 'sex'),
           sep = ' - ') %>% # actually not needed
  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

This is the most complicated form of untidy data, and typically requires a bit more massaging.

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

Think carefully about what the **observation** is for this data.

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

```
mkt_untidy %>%
    gather(day, price, starts_with('d')) %>% # wide to long
    spread(stock, price) %>% # long back to wide-ish
    mutate(day = gsub('d', '', day)) %>% # remove "d"
    unite(date, year, month, day, sep = '-') %>% # 3 cols to 1
    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
```

Tidying #3 (1/2) (using `pivot_longer` and `pivot_wider`)

```
mkt_untidy %>%
    pivot_longer(cols = starts_with('d'),
                 names_to = "day",
                 values_to = "price") %>%
    pivot_wider(names_from = stock,
                values_from = 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
```

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

```
mkt_untidy %>%
    gather(day, price, starts_with('d')) %>% # wide to long
    # spread(stock, price) %>% # long back to wide-ish
    mutate(day = gsub('d', '', day)) %>% # remove "d"
    unite(date, year, month, day, sep = '-') %>% # 3 cols to 1
    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
```

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.

First: tidy up while normalized

Then: analyze while de-normalized

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

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

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

# result table
tournament_results <- 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	year	cpi	year	cpi
2015	237	2016	240	2017	245

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

```r
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
```

Two Asides: Displaying and Organizing Data

How can we make it easier to scan raw values in a data table?

- Determine the roles of variables in your analysis (fixed by design of experiment vs. measured during course of experiment)

Displaying tidy data

How can we make it easier to scan raw values in a data table?

- Determine the roles of variables in your analysis (fixed by design of experiment vs. measured during course of experiment)
- Fixed variables should come first, then measured variables

Displaying tidy data

How can we make it easier to scan raw values in a data table?

- Determine the roles of variables in your analysis (fixed by design of experiment vs. measured during course of experiment)
- Fixed variables should come first, then measured variables
  - Order from L-R by degree of fixed-ness. The "most fixed" variables are the key describers of an observation, and are useful when we're trying to scan values.

Displaying tidy data

How can we make it easier to scan raw values in a data table?

- Determine the roles of variables in your analysis (fixed by design of experiment vs. measured during course of experiment)
- Fixed variables should come first, then measured variables
    - Order from L-R by degree of fixed-ness. The "most fixed" variables are the key describers of an observation, and are useful when we're trying to scan values.
- Put related variables next to each other

Displaying tidy data

How can we make it easier to scan raw values in a data table?

- Determine the roles of variables in your analysis (fixed by design of experiment vs. measured during course of experiment)
- Fixed variables should come first, then measured variables
    - Order from L-R by degree of fixed-ness. The "most fixed" variables are the key describers of an observation, and are useful when we're trying to scan values.
- Put related variables next to each other
- Order rows based on the first variable and then break ties with the second and subsequent (fixed) variables after that.

Organizing data in spreadsheets

Broman & Woo (2018) wrote a short paper with 12 tips for organizing data in spreadsheets for sharing, analysis, reproducibility, and collaboration. After reading the tidy data paper, I would recommend reading it.

- Be consistent
    - Codes, NA, names, ID, layout, files, dates, phrases
- Write dates like YYYY-MM-DD
- Do not leave any cells empty
- Put just one thing in a cell
- Organize the data as a single rectangle (with subjects as rows, variables as columns, and with a single header row)
- Create a data dictionary

- Do not include calculations in the raw data files
- Do not use font color or highlighting as data
- Choose good names for things
- Make backups
- Use data validation to avoid data entry errors
- Save the data in plain text files

Conclusion

Five tips to get tidy

1. Put each dataset in a table

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

Five tips to get tidy

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

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

Five tips to get tidy

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

Five tips to get tidy

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

Five tips to get tidy

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

Five tips to get tidy

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

Five tips to get tidy

1. Put each dataset in a table
   - `data.frame` (stringsAsFactors? printing to a console? lazy evaluation?)
   - `tibble` (the nice version of data.frame)
2. Put each variable in a column
3. Ask some questions
   - What are the rows of my dataset (observation, level of detail)?
   - Is each column a *distinct* variable?
   - How hard would it be to calculate a grouped aggregation?
4. Be able to describe your data set as *one row per **X** per **Y** per **Z** per* ...

Five tips to get tidy

1. Put each dataset in a table
   - `data.frame` (stringsAsFactors? printing to a console? lazy evaluation?)
   - `tibble` (the nice version of data.frame)
2. Put each variable in a column
3. Ask some questions
   - What are the rows of my dataset (observation, level of detail)?
   - Is each column a *distinct* variable?
   - How hard would it be to calculate a grouped aggregation?
4. Be able to describe your data set as *one row per **X** per **Y** per **Z** per* . . .
   - e.g. one row per iris flower, one row per household per year in which they've made a gift, one row per stock per hour per day

Five tips to get tidy

1. Put each dataset in a table
   - `data.frame` (stringsAsFactors? printing to a console? lazy evaluation?)
   - `tibble` (the nice version of data.frame)
2. Put each variable in a column
3. Ask some questions
   - What are the rows of my dataset (observation, level of detail)?
   - Is each column a *distinct* variable?
   - How hard would it be to calculate a grouped aggregation?
4. Be able to describe your data set as *one row per **X** per **Y** per **Z** per* ...
   - e.g. one row per iris flower, one row per household per year in which they've made a gift, one row per stock per hour per day
5. Structure and tidy up your data to be manipulated by a computer. Ignore urges to make it easily viewed by a human.

- Code is for *humans*, data is for *computers* (relevant tweet from Vince Buffalo)

- Code is for *humans*, data is for *computers* (relevant tweet from Vince Buffalo)
- Be consciously aware of your **values**, **variables**, and **observations**

- Code is for *humans*, data is for *computers* (relevant tweet from Vince Buffalo)
- Be consciously aware of your **values**, **variables**, and **observations**
- Normalization can be your friend

- Code is for *humans*, data is for *computers* (relevant tweet from Vince Buffalo)
- Be consciously aware of your **values**, **variables**, and **observations**
- Normalization can be your friend
- Be assertive and understanding

I consulted a number of resources for this presentation. I've arranged these in descending order of importance.

- *The* tidy data paper
- Data Organization in Spreadsheets
- Informal version of tidy data paper
- Practical introduction to tidy data
- Tidy data presentation
- Tidy data analysis (an extension of the tidy data paradigm)
- Tidy Data in Python
- Database Normalization
- Codd's 3rd Normal Form

Here's the link to my GitHub repository.

Thanks for listening!