

Data import and tidying

Week 3

AEM 2850 / 5850 : R for Business Analytics

Cornell Dyson
Spring 2023

Acknowledgements: Grant McDermott, Allison Horst, R4DS (2e)

Announcements

We will not assign any late penalties for the survey and lab-02

Late assignments will be penalized going forward

- If something comes up, please email me and Hui **in advance**

Reminders:

- Submit assignments via canvas
 - Lab-02 was due yesterday (Monday) at 11:59pm

Questions before we get started?

Plan for today

Prologue

Data import

Tidy data

- `pivot_longer`
- `pivot_wider`
- `separate`
- `unite`

Summary

Prologue

What are our concentrations?

Take a guess: what's the most common concentration among classmates?

```
## # A tibble: 42 × 2
##   program concentration
##   <chr>    <chr>
## 1 MPS      Business of Food
## 2 MPS      Bhavioral Finance
## 3 BS       Finance
## 4 BS       EERE
## 5 BS       Finance
## 6 MPS      Technology Management
## 7 MPS      Behavioral Finance
## 8 MPS      Behaviorla finance
## 9 BS       Business Analytics
## 10 BS     International Trade and Development
## # ... with 32 more rows
```

What are our concentrations?

After some processing to get concentration counts, we get:

```
## # A tibble: 1 × 10
##   ba    dev  entr enviro finance food mkting other strategy tech
##   <int> <int> <int> <int>    <int> <int> <int> <int>    <int> <int>
## 1     9      2     1     6      12      2     1     7      1      5
```

What is the "level of observation" in this data frame? (i.e., what are the rows?)

Is the best way to organize concentration counts?

How would you use counts to compute shares in this data frame?

What are our "tidy" concentrations?

Let's `pivot_longer` and `arrange` to get the top 3:

```
## # A tibble: 3 × 2
##   concentration count
##   <chr>          <int>
## 1 finance         12
## 2 ba              9
## 3 enviro          6
```

How would you use what we learned last week to compute shares?

```
tidy_concentrations |> mutate(share = count / sum(count)) # easy!
```

```
## # A tibble: 3 × 3
##   concentration count share
##   <chr>          <int> <dbl>
## 1 finance         12  0.261
## 2 ba              9   0.196
## 3 enviro          6   0.130
```

Data import

Data import

Plain-text rectangular files are a common way to store and share data:

- comma delimited files (`readr::read_csv`)
- tab delimited files (`readr::read_tsv`)
- fixed width files (`readr::read_fwf`)

We will just cover csv files since `readr` syntax is transferable

Super bowl ads: a csv example



Super bowl ads in csv format

year,brand,superbowl_ads_dot_com_url,youtube_url,funny,show_product_quickly,patriotic_celebrity,danger,animals,use_sex,id,kind,etag,view_count,like_count,dislike_count,favorite_count,comment_count,published_at,title,description,thumbnail,channel_title,category_id
2018,Toyota,<https://superbowl-ads.com/good-odds-toyota/>,<https://www.youtube.com/watch?v=zeBZvwYQ-hA>,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,zeBZvwYQ-hA,youtube#video,rn-ggKNly38Cl0C3CNjNnUH9xUw,173929,1233,38,0,NA,2018-02-03T11:29:14Z,Toyota Super Bowl Commercial 2018 Good Odds,"Toyota Super Bowl Commercial 2018 Good Odds. You can watch Toyota Super Bowl 2018 commercial. Toyota has aired its Super Bowl commercial named as Good Odds. The odds of winning a Paralympic gold medal are almost 1 billion to 1. This film follows the journey of Lauren Woolstencroft, who beat the odds to win eight Paralympic gold medals.Toyot is one of the advertiser of Super Bowl 52.

Toyota Super Bowl commercial 2018

#toyota

<https://www.youtube.com/channel/UChZGob5aWTeZreuP2hzIq7A>",<https://i.ytimg.com/vi/zeBZvwYQ-hA/sddefault.jpg>,Funny Commercials,1 2020,Bud Light,<https://superbowl-ads.com/2020-bud-light-seltzer-inside-posts-brain/>,<https://www.youtube.com/watch?v=nbbp0VW7z8w>,TRUE,TRUE,FALSE,TRUE,TRUE,FALSE,nbbp0VW7z8w,youtube#video,1roDoK-SYqSpqYwKbYrMH0jEJQ4,47752,485,14,0,14,2020-01-31T21:04:13Z,Bud Light: Post Malone #PostyStore Inside Post's Brain,"Bud Light, Post Malone ""#PostyStore Inside Post's Brain""

Garrison Sheldon (Copywriting, 2014)
Wieden + Kennedy
New York, NY

The VCU Brandcenter is a two-year, full-time master's program focused on developing the best creative problem-solvers in the world of brands. The program saw a record number of alumni ads in the 2020 Super Bowl: 26 alums contributed to 18 different spots, representing global brands like McDonald's, Facebook, Bud Light, and more. Read more here: <http://bit.ly/BCSuperBowlWatch>

Learn more about the Brandcenter: <http://www.brandcenter.vcu.edu>

Follow the Brandcenter on other social media platforms:

- Facebook (<http://www.facebook.com/VCUBrandcenter>)
- Twitter (http://www.twitter.com/VCU_Brandcenter)
- Instagram (http://www.instagram.com/vcu_brandcenter/)
- LinkedIn (<http://www.linkedin.com/school/1095073/admin/>),"<https://i.ytimg.com/vi/nbbp0VW7z8w/sddefault.jpg>,VCU Brandcenter,27 2006,Bud Light,<https://superbowl-ads.com/2006-bud-light-bear-attack/>,<https://www.youtube.com/watch?v=yk0MQD5YgV8>,TRUE, FALSE, FALSE, TRUE, TRUE, FALSE, yk0MQD5YgV8,youtube#video,0HiDfHTB3kilXfn8W0VTH0nwUIg,142310,129,15,0,9,2006-02-06T10:02:36Z,"Super Bowl 2006: Bud Light ""Save Yourself""",Bud Light Super Bowl commercial where two guys camping encounter a bear..NA,John Keehler,17

Super bowl ads in csv format

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
1	year	brand	superbowl_a	youtube_url	funny	show_produ	patriotic	celebrity	danger	animals	use_sex	id	kind	etag	view_count	like_c
2	2018	Toyota	https://supe	https://www	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	zeBZvwYQ-h	youtube#vid rn-ggKNly38	173929		
3	2020	Bud Light	https://supe	https://www	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	nbbp0VW7z	youtube#vid 1roDoK-SYqS	47752		
4	2006	Bud Light	https://supe	https://www	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	yk0MQD5Yg	youtube#vid OHiDfHTB3k	142310		
5	2018	Hynudai	https://supe	https://www	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	INPccrGk77A	youtube#vid G9Dhby9Xe1	198		
6	2003	Bud Light	https://supe	https://www	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	ovQYgnXHoc	youtube#vid acsuYsFFQ_g	13741		
7	2020	Toyota	https://supe	https://www	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE	f34Ji70u3nk	youtube#vid _AuSOSICy1	23636		
8	2020	Coca-Cola	https://supe	https://www	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	#NAME?	youtube#vid LRbAaDE85lz	304254		
9	2020	Kia	https://supe	https://www	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	IMs79UXam	youtube#vid 2zkpvY1vDy	17892		
10	2020	Hynudai	https://supe	https://www	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	WBvkmWDj	youtube#vid xs3x7jAifrHD	38385		
11	2020	Budweiser	https://supe	https://www	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	J0xugdotpp8	youtube#vid LtNOKI6-s1R	782		
12	2010	Hynudai	https://supe	https://www	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	dWQqleAYfk	youtube#vid Cr_aDDEGFI!	68458		
13	2010	Bud Light	https://supe	https://www	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE	agISXMN4tn	youtube#vid VyZVw9QGn	3805		
14	2007	Budweiser	https://supe	https://www	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	7KHIMJE84e	youtube#vid CiUp57A5ZZ	184689		
15	2002	Budweiser	https://supe	https://www	FALSE	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	1MZUvib98R	youtube#vid GrUZweFTU	121400		
16	2020	NFL	https://supe	https://www	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	lbkafMhmvl	youtube#vid GJ7PT7v2TCI	26727063		
17	2017	NFL	https://supe	https://www	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	9csH_Hy-7A	youtube#vid 5ltssVIMP6r	865781		
18	2005	Bud Light	https://supe	https://www	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	#NAME?	youtube#vid 21rlALfdTQ	8990		
19	2009	Pepsi	https://supe	https://www	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	qUZASe77ig	youtube#vid N1MkFXKysC	129399		
20	2004	Bud Light	https://supe	https://www	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	mjn3OY9Eef	youtube#vid xGxW4TQM	3739		
21	2020	Doritos	https://supe	https://www	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE	6xVnq31Vk8	youtube#vid W8qRERQW	1990447		
22	2007	Bud Light	https://supe	https://www	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	aOfRpCp6Pg	youtube#vid yMYg7XcZg0	72997		
23	2020	Pepsi	https://supe	https://www	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	Ozb_fJB_z6o	youtube#vid llag9pAYBM	29219		
24	2019	Hynudai	https://supe	https://www	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	6mfJ0EsqLW	youtube#vid sGuPl2nEq3e	38968		

Getting from a csv to a data frame

How are data frames and csv files similar?

- both are rectangular
- csv lines often correspond to rows
- csv commas delineate columns

How are they different?

- csv files do not store column types!

1) `readr::read_csv`

`read_csv` helps us get from point a to point b:

- you give it the path to your csv file
- it takes the first line of data as column names by default
- it guesses column types and builds up a data frame

Most csv files can be read using the defaults, so we will focus on that

If you run into special cases, consult `?read_csv`

An aside on paths

First we need to figure out what path to give `read_csv`

We have several options:

- absolute paths
 - `/Users/todd/aem2850/slides/data/superbowl.csv`
 - `C:\aem2850\slides\data\superbowl.csv`
- relative paths
 - `data/superbowl.csv`
- links
 - `https://raw.githubusercontent.com/superbowl-ads.csv`
- others
 - see `?read_csv`

An aside on paths

Relative paths are nice: work if you move R code to another directory or device

But what are they relative to?

The working directory

But where is the working directory?

R scripts (.R) in an R Project: default working directory is the project directory

Quarto (.qmd): default working directory is the location of the .qmd file

In our RStudio Cloud projects, these will both be **/cloud/project**, which is the default directory for our projects and where we store our .qmd templates

An aside on paths

More generally, can check your working directory using `getwd()`

For example, here's the working directory for the slides' source code:

```
getwd()
```

```
## [1] "/Users/todd/Documents/GitHub/aem2850/content/slides"
```

`setwd()` will alter the working directory

An aside on paths

Just for context, here is a list of (some) directories below the working directory

```
## .
## └── css
## └── data
##   └── img
##   └── libs
```

Now list (some of) the .csv files in the folder **data**:

```
## data
## └── superbowl.csv
```

data/superbowl.csv is the relative path from our working directory to our csv

1) `readr::read_csv`

So all we need to do is give that relative path to `read_csv`:

```
ads <- read_csv("data/superbowl.csv")

## #> #> Rows: 247 Columns: 25
## #> #> — Column specification ——————
## #> #> Delimiter: ","
## #> #> chr (10): brand, superbowl_ads_dot_com_url, youtube_url, id, kind, etag, ti...
## #> #> dbl (7): year, view_count, like_count, dislike_count, favorite_count, comm...
## #> #> lgl (7): funny, show_product_quickly, patriotic, celebrity, danger, animal...
## #> #> dttm (1): published_at
## #>
## #> #> i Use `spec()` to retrieve the full column specification for this data.
## #> #> i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

That was easy!

Now we're back to our old tricks

```
ads |>  
  count(funny) |>  
  arrange(desc(n))
```

```
## # A tibble: 2 × 2  
##   funny     n  
##   <lgl> <int>  
## 1 TRUE     171  
## 2 FALSE     76
```

```
ads |>  
  group_by(brand) |>  
  summarize(likes = sum(like_count, na.rm = T)) |>  
  arrange(desc(likes))
```

```
## # A tibble: 10 × 2  
##   brand      likes  
##   <chr>     <dbl>  
## 1 Doritos    326151  
## 2 NFL        224263  
## 3 Coca-Cola  160231  
## 4 Bud Light   104380  
## 5 Budweiser   88765  
## 6 Pepsi       14796  
## 7 Toyota      5316  
## 8 Hynudai     4204  
## 9 E-Trade     2624  
## 10 Kia         2127
```

2) `readr::write_csv`

Use `write_csv` to write data to a `.csv`:

```
write_csv(ads, "data/superbowl.csv") # overwrite the raw data (bad idea!)  
ads |>  
  select(year, brand, youtube_url) |>  
  write_csv("superbowl-urls.csv") # write modified output to a new file
```

csv is not always the best storage medium

To preserve column specifications and save time when working in R, use `write_rds()` and `read_rds()` to save and open data frames in .RDS format

For multiple objects, `save()` and `load()` are handy functions from base R that work with the .RData format

Importing excel data

Wait, what about getting data from excel spreadsheets?

Use `readxl:::read_excel()` for data that is stored in `.xls` or `.xlsx` format

`readxl` isn't loaded as part of the core tidyverse, so we need to load it first:

```
library(readxl) # already installed as part of the tidyverse, but not loaded by default  
excel_sheets("data/readxl/datasets.xlsx") # list the sheets in datasets.xlsx
```

```
## [1] "iris"      "mtcars"     "chickwts"   "quakes"
```

Importing excel data

```
read_excel("data/readxl/datasets.xlsx", sheet = "mtcars")
```

```
## # A tibble: 32 × 11
##       mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##     <dbl> <dbl>
## 1     21      6   160   110   3.9   2.62  16.5     0     1     4     4
## 2     21      6   160   110   3.9   2.88  17.0     0     1     4     4
## 3    22.8     4   108    93   3.85   2.32  18.6     1     1     4     1
## 4    21.4     6   258   110   3.08   3.22  19.4     1     0     3     1
## 5    18.7     8   360   175   3.15   3.44  17.0     0     0     3     2
## 6    18.1     6   225   105   2.76   3.46  20.2     1     0     3     1
## 7    14.3     8   360   245   3.21   3.57  15.8     0     0     3     4
## 8    24.4     4   147.    62   3.69   3.19   20        1     0     4     2
## 9    22.8     4   141.    95   3.92   3.15  22.9     1     0     4     2
## 10   19.2     6   168.   123   3.92   3.44  18.3     1     0     4     4
## # ... with 22 more rows
```

Importing excel data

Can also use arguments like `range` to get data using excel lingo

```
read_excel("data/readxl/datasets.xlsx",
           range = "mtcars!A1:D6") # note the combination of sheet and cell range
```

```
## # A tibble: 5 × 4
##      mpg   cyl  disp    hp
##      <dbl> <dbl> <dbl> <dbl>
## 1    21     6   160   110
## 2    21     6   160   110
## 3  22.8     4   108    93
## 4  21.4     6   258   110
## 5  18.7     8   360   175
```

Tidy data

WHAT is "tidy" data?

“**TIDY DATA** is a standard way of mapping the meaning of a dataset to its structure.”

—HADLEY WICKHAM

In tidy data:

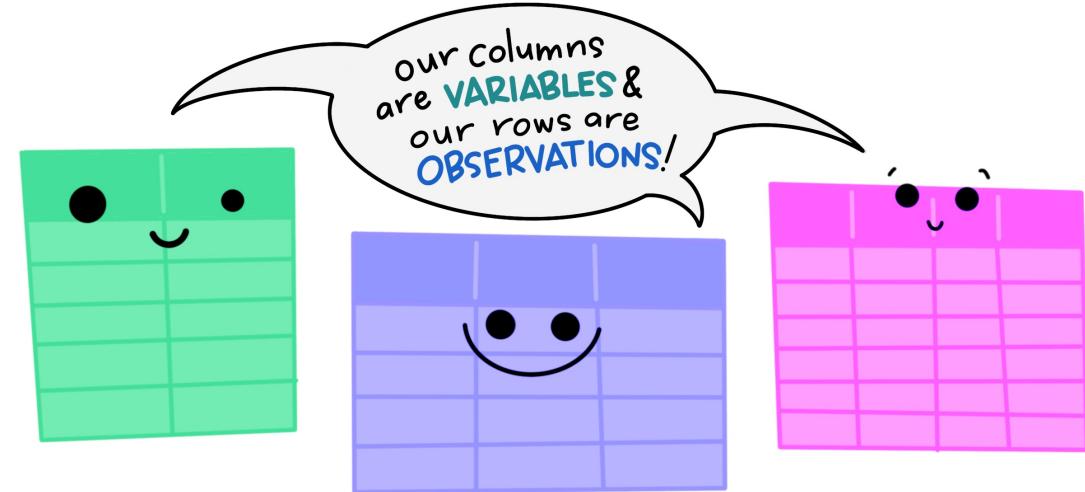
- each variable forms a column
- each observation forms a row
- each cell is a single measurement

each column a variable

each row an observation

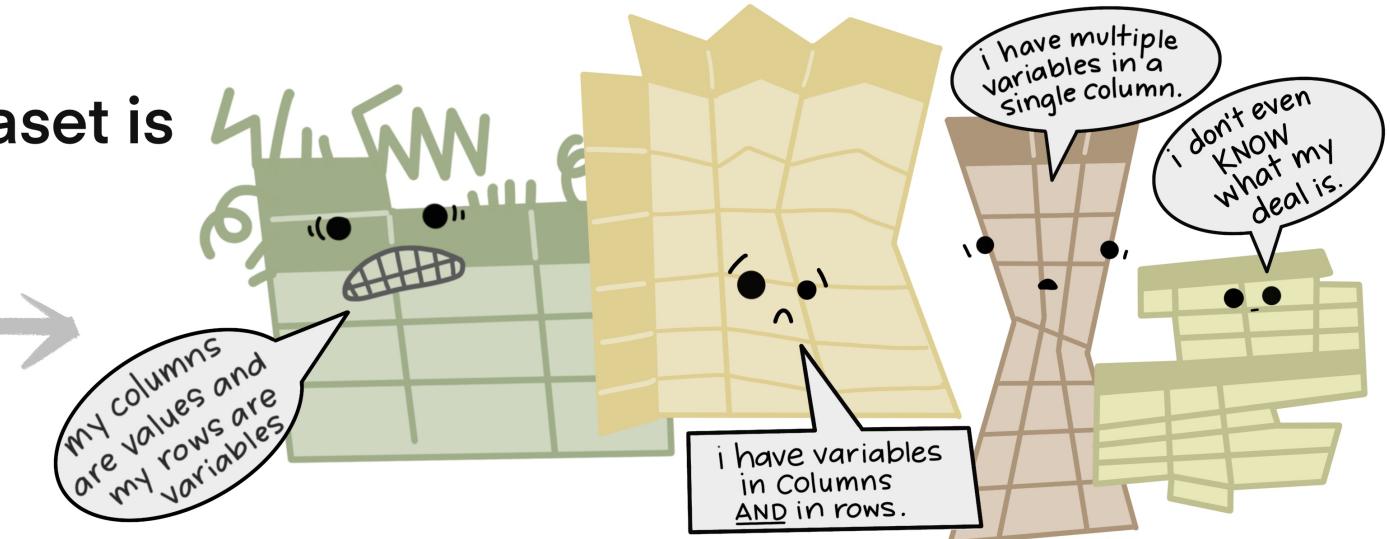
id	name	color
1	floof	gray
2	max	black
3	cat	orange
4	donut	gray
5	merlin	black
6	panda	calico

The standard structure of
tidy data means that
“tidy datasets are all alike...”



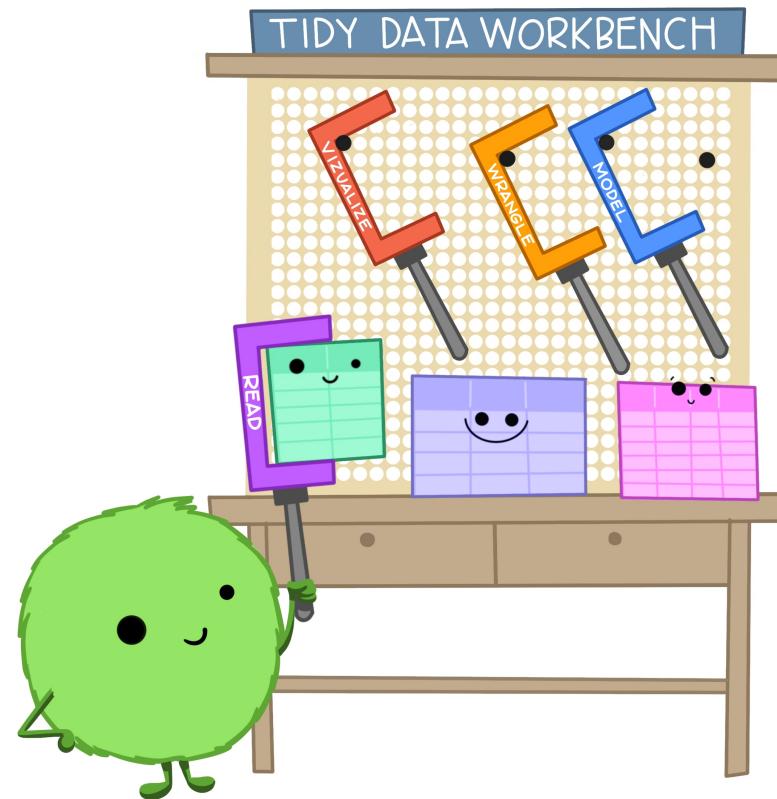
“...but every messy dataset is
messy in its own way.”

-HADLEY WICKHAM

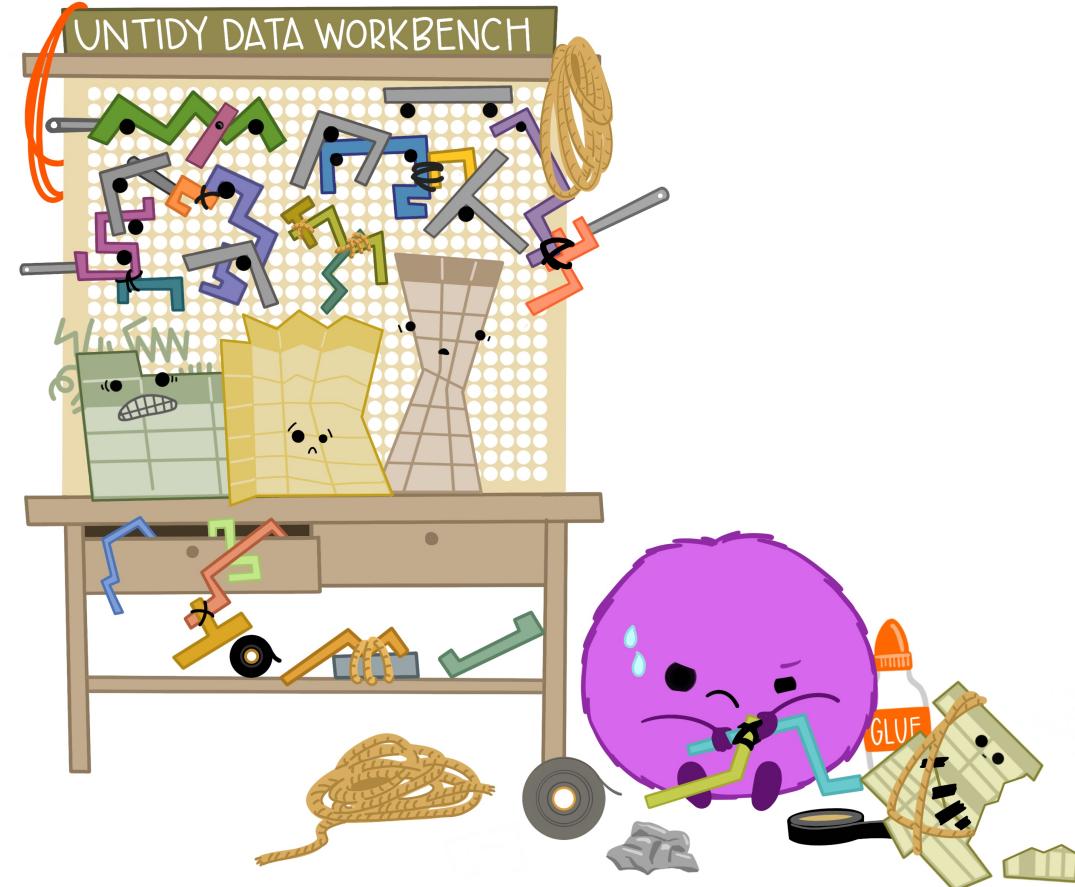


WHY "tidy" data?

When working with tidy data,
we can use the **same tools** in
similar ways for different datasets...



...but working with untidy data often means
reinventing the wheel with **one-time**
approaches that are hard to iterate or reuse.



Which of these do you think is most tidy?

table1

```
#> # A tibble: 4 × 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
```

table3

```
#> # A tibble: 8 × 4
#>   country     year   type     count
#>   <chr>      <dbl>   <chr>     <dbl>
#> 1 Afghanistan 1999 cases      745
#> 2 Afghanistan 1999 population 19987071
#> 3 Afghanistan 2000 cases     2666
#> 4 Afghanistan 2000 population 20595360
#> 5 Brazil      1999 cases      37737
#> 6 Brazil      1999 population 172006362
#> # ... with 2 more rows
```

table2

```
#> # A tibble: 4 × 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
```

Here, **table1**. Though in general, it can depend on your objectives!

Why is `table1` a good choice?

The `dplyr` functions from last week are designed to work with tidy data:

```
# compute rate per 10,000
table1 |>
  mutate(rate = cases / population * 10000)

## # A tibble: 4 × 5
##   country     year cases population    rate
##   <chr>      <dbl> <dbl>       <dbl> <dbl>
## 1 Afghanistan 1999    745 19987071 0.373
## 2 Afghanistan 2000   2666 20595360 1.29 
## 3 Brazil       1999 37737 172006362 2.19 
## 4 Brazil       2000 80488 174504898 4.61
```

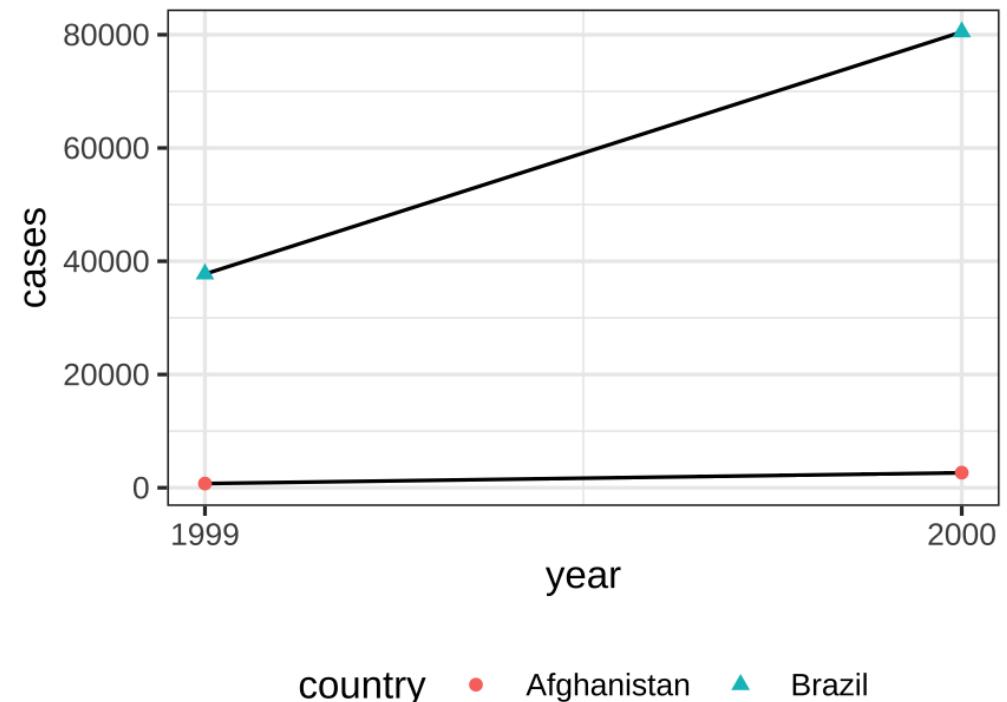
```
# compute cases per year
table1 |>
  group_by(year) |>
  summarize(cases = sum(cases))

## # A tibble: 2 × 2
##   year cases
##   <dbl> <dbl>
## 1 1999 38482
## 2 2000 83154
```

Why is `table1` a good choice?

The `ggplot2` functions we will learn are designed to work with tidy data:

```
# visualize changes over time
table1 |>
  ggplot(aes(x = year, y = cases)) +
  geom_line(aes(group = country)) +
  geom_point(aes(color = country,
                  shape = country)) +
  scale_x_continuous(breaks = c(1999, 2000)) +
  theme_bw() +
  theme(legend.position = "bottom")
```



HOW can we "tidy" data?

Key `tidyverse` verbs

1. `pivot_longer`: Pivot wide data into long format (updated version of `gather`)
2. `pivot_wider`: Pivot long data into wide format (updated version of `spread`)
3. `separate`: Split one column into multiple columns
4. `unite`: Combine multiple columns into one

Let's start with an untidy dataset

```
stocks
```

```
## # A tibble: 2 × 4
##   date           X     Y     Z
##   <date>     <dbl> <dbl> <dbl>
## 1 2021-01-01  0.586 -0.219  2.42
## 2 2021-01-02  0.709 -0.907 -7.27
```

We have 4 columns, the date and the stocks

Is this tidy?

No!

1. stock names **X, Y, Z** are values, not variables
2. cells contain prices, but there is no variable **price**

1) `tidyverse::pivot_longer`

We need to pivot the stock name columns X, Y, Z "longer"

We need to give three key pieces of information to `pivot_longer()`:

1. Choose columns using `X:Z` or `-date`
2. Decide what variable holds the names: `names_to = "stock"`
3. Decide what variable holds the values: `values_to = "price"`

Here is what that syntax looks like:

```
stocks |>  
  pivot_longer(X:Z, names_to = "stock", values_to = "price")
```

1) `tidyr::pivot_longer`

Here is what that syntax does:

```
stocks |> pivot_longer(X:Z, names_to = "stock", values_to = "price")
```

```
## # A tibble: 6 × 3
##   date      stock  price
##   <date>    <chr>  <dbl>
## 1 2021-01-01 X      0.586
## 2 2021-01-01 Y     -0.219
## 3 2021-01-01 Z      2.42 
## 4 2021-01-02 X      0.709
## 5 2021-01-02 Y     -0.907
## 6 2021-01-02 Z     -7.27
```

Let's save the "tidy" (i.e., long) stocks data frame for later use

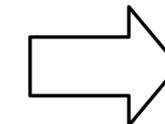
```
tidy_stocks <- stocks |>
  pivot_longer(-date, names_to = "stock", values_to = "price")
```

1) tidyverse::pivot_longer: How does it work?

```
df |>  
  pivot_longer(  
    cols = col1:col2,  
    names_to = "name",  
    values_to = "value"  
  )
```

Existing variables such as **var** need to be repeated, once for each column in **cols**

var	col1	col2
A	1	2
B	3	4
C	5	6



var	name	value
A	col1	1
A	col2	2
B	col1	3
B	col2	4
C	col1	5
C	col2	6

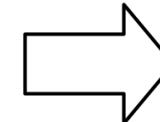
1) tidyverse::pivot_longer: How does it work?

```
df |>  
  pivot_longer(  
    cols = col1:col2,  
    names_to = "name",  
    values_to = "value"  
  )
```

Column names become
values in a new variable,
whose name is given by
`names_to`

- repeated once for each row in the original dataset

var	col1	col2
A	1	2
B	3	4
C	5	6



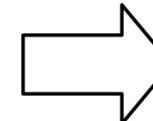
var	name	value
A	col1	1
A	col2	2
B	col1	3
B	col2	4
C	col1	5
C	col2	6

1) tidyverse::pivot_longer: How does it work?

```
df |>  
  pivot_longer(  
    cols = col1:col2,  
    names_to = "name",  
    values_to = "value"  
  )
```

Cell values become values in
a new variable, with a name
given by `values_to`

var	col1	col2
A	1	2
B	3	4
C	5	6



var	name	value
A	col1	1
A	col2	2
B	col1	3
B	col2	4
C	col1	5
C	col2	6

2) tidy::pivot_wider

```
tidy_stocks |> pivot_wider(names_from = stock, values_from = price)
```

```
## # A tibble: 2 × 4
##   date           X       Y       Z
##   <date>     <dbl>   <dbl>   <dbl>
## 1 2021-01-01  0.586 -0.219  2.42
## 2 2021-01-02  0.709 -0.907 -7.27
```

We just got our original data back!

```
stocks
```

```
## # A tibble: 2 × 4
##   date           X       Y       Z
##   <date>     <dbl>   <dbl>   <dbl>
## 1 2021-01-01  0.586 -0.219  2.42
## 2 2021-01-02  0.709 -0.907 -7.27
```

2) tidyverse::pivot_wider

```
tidy_stocks |> pivot_wider(names_from = date, values_from = price)
```

```
## # A tibble: 3 × 3
##   stock `2021-01-01` `2021-01-02`
##   <chr>     <dbl>     <dbl>
## 1 X         0.586     0.709
## 2 Y        -0.219    -0.907
## 3 Z         2.42      -7.27
```

Reversing the arguments effectively transposed the original data

```
stocks
```

```
## # A tibble: 2 × 4
##   date          X          Y          Z
##   <date>      <dbl>      <dbl>      <dbl>
## 1 2021-01-01  0.586    -0.219     2.42
## 2 2021-01-02  0.709    -0.907    -7.27
```

3) `tidy::separate`

```
halftime_openers <- data.frame(name = c("Dr.Dre", "Snoop.Dogg")) # (from the 2022 super bowl)
halftime_openers
```

```
##           name
## 1      Dr.Dre
## 2 Snoop.Dogg
```

```
halftime_openers |> separate(name, c("first_name", "last_name"))
```

```
##   first_name last_name
## 1        Dr       Dre
## 2     Snoop     Dogg
```

This command is clever: it splits at non-alphanumeric values. To avoid ambiguity, you can specify the separation character with `separate(..., sep=".")`

3) tidyverse::separate

```
halftime_performers <- data.frame(name = c("Dr..Dre", "Snoop.Dogg", "Eminem", "Mary.J.Blige",  
"Kendrick.Lamar", "50.Cent", "Anderson..Paak"))
```

```
halftime_performers |> separate(name, c("first_name", "last_name"))
```

```
## Warning: Expected 2 pieces. Additional pieces discarded in 1 rows [4].
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 1 rows [3].
```

```
##   first_name last_name  
## 1        Dr      Dre  
## 2     Snoop     Dogg  
## 3    Eminem     <NA>  
## 4      Mary       J  
## 5  Kendrick    Lamar  
## 6        50     Cent  
## 7    Anderson     Paak
```

What happened? Did we lose any information?

4) `tidyverse::unite`

Sometimes a single variable is stored in multiple columns:

```
gas_prices
```

```
##      yr mnth dy price
## 1 2023    2  3 3.567
## 2 2023    2  4 3.565
## 3 2023    2  5 3.563
## 4 2023    2  6 3.559
```

4) tidyverse::unite

```
# combine "yr", "mnth", and "dy" into one "date" column
gp_u <- gas_prices |> unite(date, c("yr", "mnth", "dy"), sep = "-")
gp_u
```

```
##           date price
## 1 2023-2-3 3.567
## 2 2023-2-4 3.565
## 3 2023-2-5 3.563
## 4 2023-2-6 3.559
```

4) `tidyr::unite`

Note that `unite` will automatically create a character variable:

```
gp_u |> as_tibble() # print it as a tibble to see data types
```

```
## # A tibble: 4 × 2
##   date      price
##   <chr>    <dbl>
## 1 2023-2-3  3.57
## 2 2023-2-4  3.56
## 3 2023-2-5  3.56
## 4 2023-2-6  3.56
```

To convert it to something else (e.g., date or numeric), modify it using `mutate`

4) tidyverse::unite

```
library(lubridate) # the tidyverse package lubridate is helpful for dates and times
gp_u |>
  mutate(date = ymd(date)) |> # ymd() converts from strings to dates, more on that later
  as_tibble()
```

```
## # A tibble: 4 × 2
##   date      price
##   <date>    <dbl>
## 1 2023-02-03  3.57
## 2 2023-02-04  3.56
## 3 2023-02-05  3.56
## 4 2023-02-06  3.56
```

tidy resources

Data tidying with `tidy` :: CHEAT SHEET (link downloads a pdf)

Vignette (from the `tidy` package)

```
vignette("tidy-data")
```

Original paper (Hadley Wickham, 2014 JSS)

Summary

Key verbs so far

Import

readr

1. `read_csv`
2. `write_csv`

readxl

1. `read_excel`

Tidy

tidyverse

1. `pivot_longer`
2. `pivot_wider`
3. `separate`
4. `unite`

Transform

dplyr

1. `filter`
2. `arrange`
3. `select`
4. `mutate`
5. `summarize`