Data import and tidying

Week 3

AEM 2850 / 5850 : R for Business Analytics
Cornell Dyson
Fall 2025

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

Announcements

Submit assignments via gradescope

- Either via canvas or directly in gradescope
- Homework - Week 2 was due yesterday (Monday) at 11:59pm

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

Late assignments will be penalized going forward

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

Questions before we get started?

Plan for this week

Tuesday

Prologue

Data import

- read_csv
- write_csv
- read_excel

example-03-1

Thursday

Tidy data

- pivot_longer
- pivot_wider

Summary

example-03-2

Prologue

What are our concentrations?

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

Let's figure this out using the course survey

First, let's **import** the data:

```
responses <- read_csv("data/survey-responses/homework-1-survey.csv")
```

```
## Rows: 114 Columns: 22
## ── Column specification ────────────────────────────────────────────────
## Delimiter: ","
## chr (22): Timestamp, Email Address, Name, netid, Name pronunciation guide (o...
##
## 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.
```

What are our concentrations?

```
responses |>
  select(Concentration)
```

```
## # A tibble: 114 × 1
##    Concentration
##    <chr>
##  1 Finance
##  2 Finance
##  3 Business Analytics and Accounting
##  4 Business Analytics
##  5 Entrepreneurship
##  6 Entrepreneurship
##  7 Finance
##  8 Strategy
##  9 Business Analytics
## 10 Business Analytics
## # ℹ 104 more rows
```

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

What are our concentrations?

After some processing to get concentration counts, we get:

```
## # A tibble: 1 × 12
##    acct    ba   dev  econ  entr enviro finance  food  mgmt  mktg other strategy
##   <int> <int> <int> <int> <int>  <int>   <int> <int> <int> <int> <int>    <int>
## 1     6    20     5     7     6      3      39    10     3     9     3       10
```

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 `slice_max` to get the top 3:

```
## # A tibble: 4 × 2
##   concentration count
##   <chr>         <int>
## 1 finance          39
## 2 ba               20
## 3 food             10
## 4 strategy         10
```

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

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

```
## # A tibble: 4 × 3
##   concentration count  share
##   <chr>         <int>  <dbl>
## 1 finance          39 0.322
## 2 ba               20 0.165
## 3 food             10 0.0826
## 4 strategy         10 0.0826
```

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

superbowl.csv — Edited

```
year,brand,superbowl_ads_dot_com_url,youtube_url,funny,show_product_quickly,patriotic,celebrity,d
anger,animals,use_sex,id,kind,etag,view_count,like_count,dislike_count,favorite_count,comment_cou
nt,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,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""
```

Super bowl ads in csv format

	A	B	C	D	E	F	G	H	I	J	K	L	
1	year	brand	superbowl_a	youtube_url	funny	show_produ	patriotic	celebrity	danger	animals	use_sex	id	kin
2	2018	Toyota	https://supe	https://www	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	zeBZvwYQ-h/	yo
3	2020	Bud Light	https://supe	https://www	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	nbbp0VW7z8	yo
4	2006	Bud Light	https://supe	https://www	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	ykOMQD5Yg\	yo
5	2018	Hynudai	https://supe	https://www	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	lNPccrGk77A	yo
6	2003	Bud Light	https://supe	https://www	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	ovQYgnXHoo	yo
7	2020	Toyota	https://supe	https://www	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE	f34Ji70u3nk	yo
8	2020	Coca-Cola	https://supe	https://www	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	#NAME?	yo
9	2020	Kia	https://supe	https://www	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	lMs79UXam9	yo
10	2020	Hynudai	https://supe	https://www	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	WBvkmWDjs	yo
11	2020	Budweiser	https://supe	https://www	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	J0xugdotpp8	yo
12	2010	Hynudai	https://supe	https://www	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	dWQqIeAYfK	yo
13	2010	Bud Light	https://supe	https://www	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE	agISXMN4tng	yo
14	2007	Budweiser	https://supe	https://www	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	7KHlMJE84e/	yo
15	2002	Budweiser	https://supe	https://www	FALSE	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	1MZUvib98R	yo
16	2020	NFL	https://supe	https://www	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	lbkafMhmvM	yo
17	2017	NFL	https://supe	https://www	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	9csH_Hy-7A4	yo
18	2005	Bud Light	https://supe	https://www	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	#NAME?	yo

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:

1. you give it the path to your csv file
2. it takes the first line of data as column names by default
3. 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

We need to figure out what "path" to give `read_csv` to get to our file

The simplest thing is to provide a "relative path" from 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

1) readr::read_csv

Here is an example for the file `superbowl.csv` inside the folder `data`:

```
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

```r
ads |>
  count(funny)
```

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

```r
ads |>
  group_by(brand) |>
  summarize(likes = sum(like_count, na.rm = TRUE)) |>
  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`:

```
# overwrite the raw data (bad idea!)
ads |>
  write_csv("data/superbowl.csv")
```

```
# modify the data and then output it to a new file
ads |>
  select(year, brand, youtube_url) |>
  write_csv("superbowl-urls.csv")
```

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

```
## # A tibble: 6 × 11
##     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <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
```

example-03-1:
import-data-practice.R

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

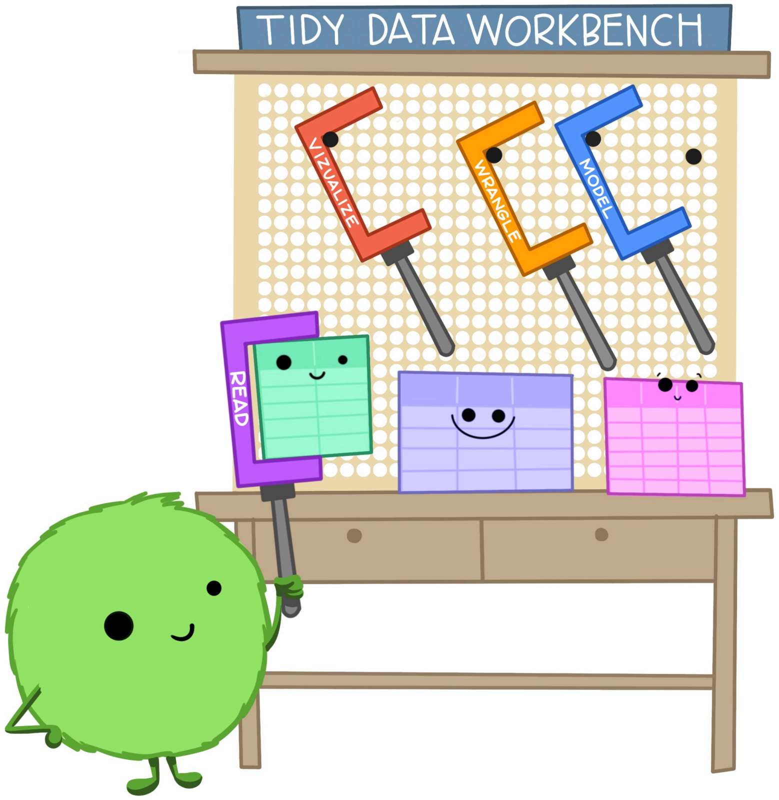
id	name	color
1	floof	gray
2	max	black
3	cat	orange
4	donut	gray
5	merlin	black
6	panda	calico

each row an observation

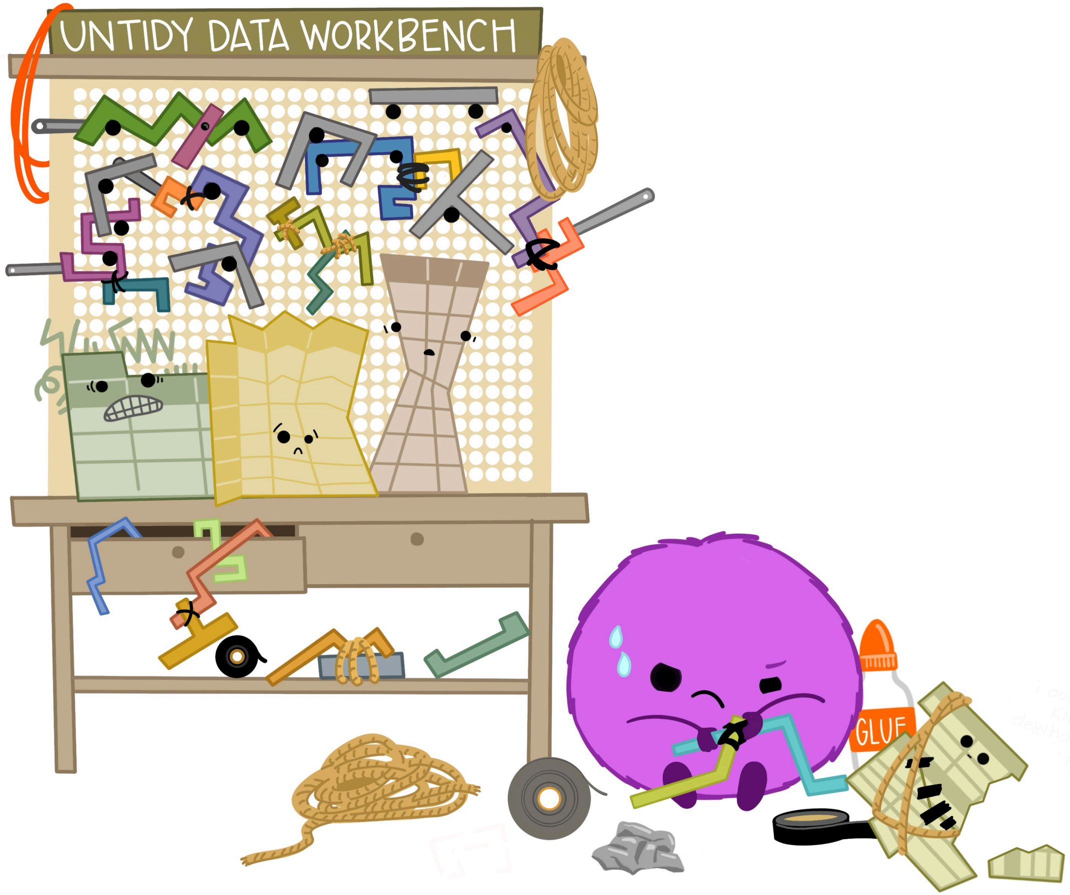
Wickham, H. (2014). Tidy Data. Journal of Statistical Software *59* (10). DOI: 10.18637/jss.v059.i10

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
```

```
table2
#> # 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
```

```
table3
#> # 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 people
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
```

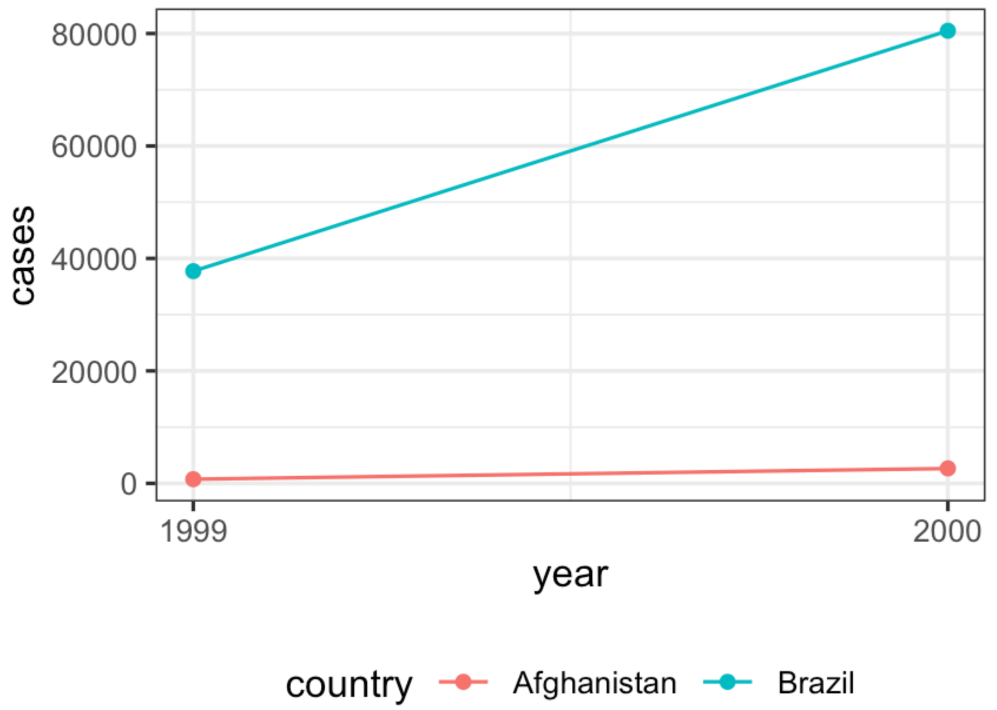
```
# compute cases per year
table1 |>
  group_by(year) |>
  summarize(cases = sum(cases))
```

```
## # 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:

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



How can we "tidy" data?

Key tidyr verbs

1. `pivot_longer`: Pivot wide data into long format

2. `pivot_wider`: Pivot long data into wide format

Let's start with an untidy dataset

```
stocks
```

```
## # A tibble: 2 × 4
##   date        AAPL  GOOG  MSFT
##   <date>     <dbl> <dbl> <dbl>
## 1 2025-09-15  234.  199.  412.
## 2 2025-09-14  236.  195.  364.
```

We have 4 columns, the date and the stocks

Why do I think this is untidy?

1. stock names `AAPL`, `GOOG`, `MSFT` are values, not variables
2. cells contain prices, but there is no variable `price`

1) tidyr::pivot_longer

Let's use `pivot_longer()` to get this in tidy form

We want to pivot the stock name columns `AAPL`, `GOOG`, `MSFT` "longer"

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

1. **Which columns to pivot** using `c(AAPL, GOOG, MSFT)` or `-date`
2. **What variable will hold the names** using `names_to = "stock"`
3. **What variable will hold the values** using `values_to = "price"`

Here is what that syntax looks like:

```
stocks |>
  pivot_longer(cols = c(AAPL, GOOG, MSFT), names_to = "stock", values_to = "price")
```

1) tidyr::pivot_longer

Here is what that syntax does:

```
stocks |> pivot_longer(cols = c(AAPL, GOOG, MSFT), names_to = "stock", values_to = "price")
```

```
## # A tibble: 6 × 3
##   date       stock price
##   <date>     <chr> <dbl>
## 1 2025-09-15 AAPL   234.
## 2 2025-09-15 GOOG   199.
## 3 2025-09-15 MSFT   412.
## 4 2025-09-14 AAPL   236.
## 5 2025-09-14 GOOG   195.
## 6 2025-09-14 MSFT   364.
```

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

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

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

```r
df |>
  pivot_longer(
    cols = c(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) tidyr::pivot_longer: How does it work?

```
df |>
  pivot_longer(
    cols = c(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

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) tidyr::pivot_longer: How does it work?

```
df |>
  pivot_longer(
    cols = c(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) tidyr::pivot_wider

We can use `pivot_wider()` to make data "wide"

Let's say we want to convert `tidy_stocks` back to `stocks`

```
tidy_stocks
```

```
## # A tibble: 6 × 3
##   date       stock price
##   <date>     <chr> <dbl>
## 1 2025-09-15 AAPL   234.
## 2 2025-09-15 GOOG   199.
## 3 2025-09-15 MSFT   412.
## 4 2025-09-14 AAPL   236.
## 5 2025-09-14 GOOG   195.
## 6 2025-09-14 MSFT   364.
```

```
stocks
```

```
## # A tibble: 2 × 4
##   date        AAPL  GOOG  MSFT
##   <date>     <dbl> <dbl> <dbl>
## 1 2025-09-15  234.  199.  412.
## 2 2025-09-14  236.  195.  364.
```

2) tidyr::pivot_wider

```
tidy_stocks
```

```
## # A tibble: 6 × 3
##   date       stock price
##   <date>     <chr> <dbl>
## 1 2025-09-15 AAPL   234.
## 2 2025-09-15 GOOG   199.
## 3 2025-09-15 MSFT   412.
## 4 2025-09-14 AAPL   236.
## 5 2025-09-14 GOOG   195.
## 6 2025-09-14 MSFT   364.
```

We need to give two key pieces of information to `pivot_wider()`:

1. **What variable to get names from:** `names_from = stock`
2. **What variable to get values from:** `values_from = price`

`pivot_wider()` figures out the rest

2) tidyr::pivot_wider

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

```
## # A tibble: 2 × 4
##   date        AAPL  GOOG  MSFT
##   <date>     <dbl> <dbl> <dbl>
## 1 2025-09-15  234.  199.  412.
## 2 2025-09-14  236.  195.  364.
```

We just got our original data back!

```
stocks
```

```
## # A tibble: 2 × 4
##   date        AAPL  GOOG  MSFT
##   <date>     <dbl> <dbl> <dbl>
## 1 2025-09-15  234.  199.  412.
## 2 2025-09-14  236.  195.  364.
```

2) tidyr::pivot_wider

Reversing the arguments effectively transposes the original data:

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

```
## # A tibble: 3 × 3
##   stock `2025-09-15` `2025-09-14`
##   <chr>        <dbl>        <dbl>
## 1 AAPL          234.         236.
## 2 GOOG          199.         195.
## 3 MSFT          412.         364.
```

```
stocks
```

```
## # A tibble: 2 × 4
##   date        AAPL  GOOG  MSFT
##   <date>     <dbl> <dbl> <dbl>
## 1 2025-09-15  234.  199.  412.
## 2 2025-09-14  236.  195.  364.
```

tidyr resources

Data tidying with tidyr : : CHEATSHEET

Vignette (from the **tidyr** package)

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

Ch. 5 of R for Data Science (2e)

Original paper (Hadley Wickham, 2014 JSS)

Summary

Key verbs so far

Import

readr

1. `read_csv`
2. `write_csv`

readxl

1. `read_excel`

Tidy

tidyr

1. `pivot_longer`
2. `pivot_wider`

Transform

dplyr

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

example-03-2:
tidy-data-practice.R