

Data import, tidy data, and relational data

Week 4

AEM 2850: R for Business Analytics
Cornell Dyson
Spring 2022

Acknowledgements: Grant McDermott, Jenny Bryan, Allison Horst

Announcements

Late assignments will get a ✓- going forward

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

Reminders:

- Submit assignments via canvas
 - Lab 3 was due yesterday (Monday) at 11:59pm
 - Reflection - Week 4 is due Wednesday at 11:59pm
- Bring your laptop to class on Thursday

Questions before we get started?

Plan for today

Prologue

Data import

Tidy data

Relational data

Summary

Prologue

What are our concentrations?

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

```
## # A tibble: 39 × 1
##   value
##   <chr>
## 1 Business Analytics (AEM) Interactive Technologies (IS)
## 2 Finance
## 3 Business Analytics
## 4 Data Science and Finance
## 5 Sustainable Business and Economic Policy
## 6 Finance
## 7 Finance
## 8 Business Analytics and Strategy
## 9 Finance, Business Analytics
## 10 Finance
## # ... with 29 more rows
```

What are our concentrations?

After some processing to get concentration counts, we get:

```
## # A tibble: 1 × 7
##   accounting    ba    entr enviro finance food strategy
##   <int>     <int> <int>   <int>   <int> <int>    <int>
## 1       2      21      2      3      19      2       4
```

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 ba                 21
## 2 finance            19
## 3 strategy            4
```

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

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

```
## # A tibble: 3 × 3
##   concentration count  share
##   <chr>           <int>  <dbl>
## 1 ba                 21  0.396
## 2 finance            19  0.358
## 3 strategy            4  0.0755
```

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

Super bowl ads in csv format

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`

1) `readr::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 %>%
  group_by(funny) %>%
  count() %>%
  arrange(desc(n))
```

```
## # A tibble: 2 × 2
## # Groups:   funny [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)
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:

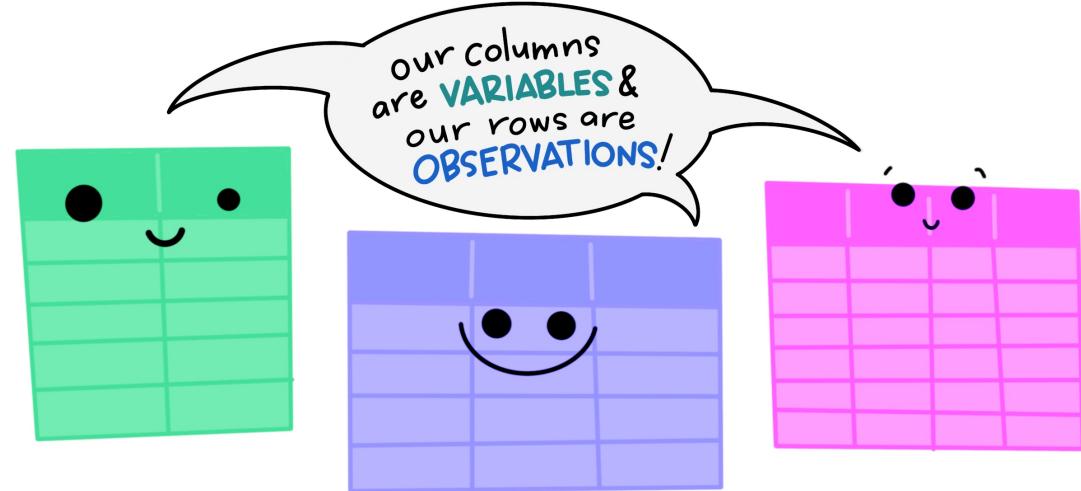
- each variable forms a column
- each observation forms a row
- each cell is a single measurement

each column a variable

each row an observation

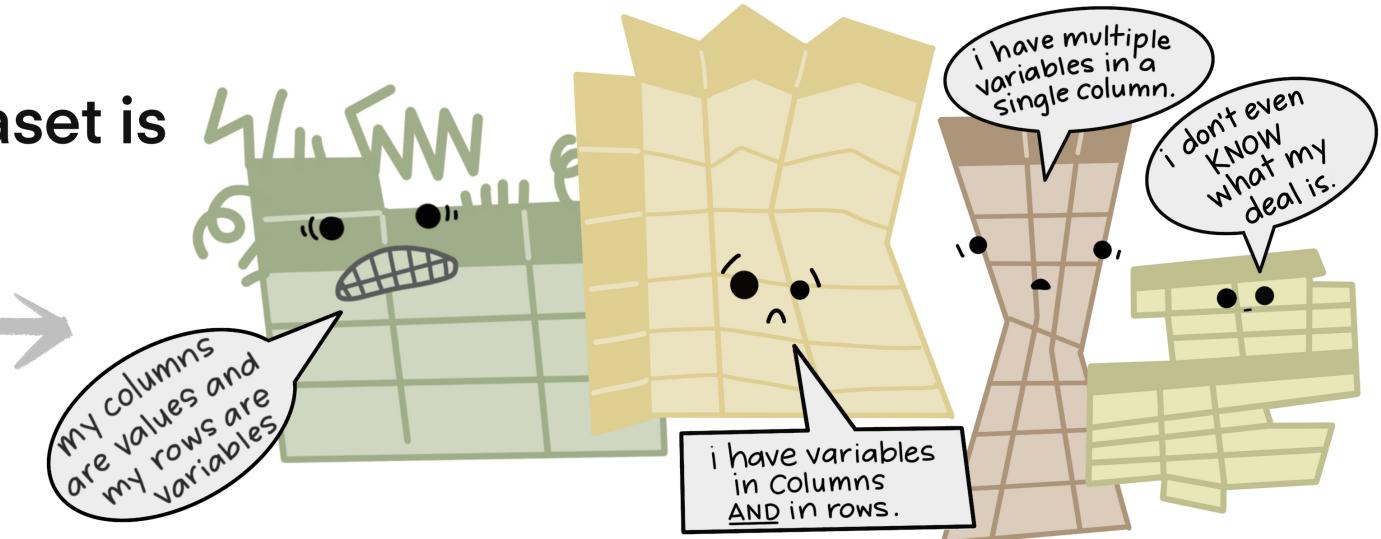
id	name	color
1	floof	gray
2	max	black
3	cat	orange
4	donut	gray
5	merlin	black
6	panda	calico

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



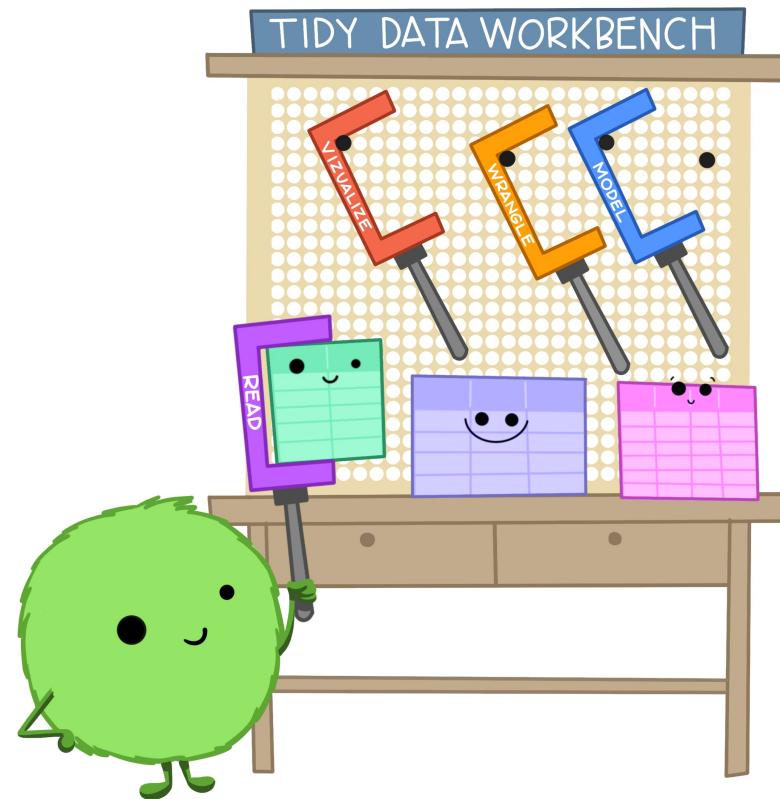
“...but every messy dataset is
messy in its own way.”

-HADLEY WICKHAM

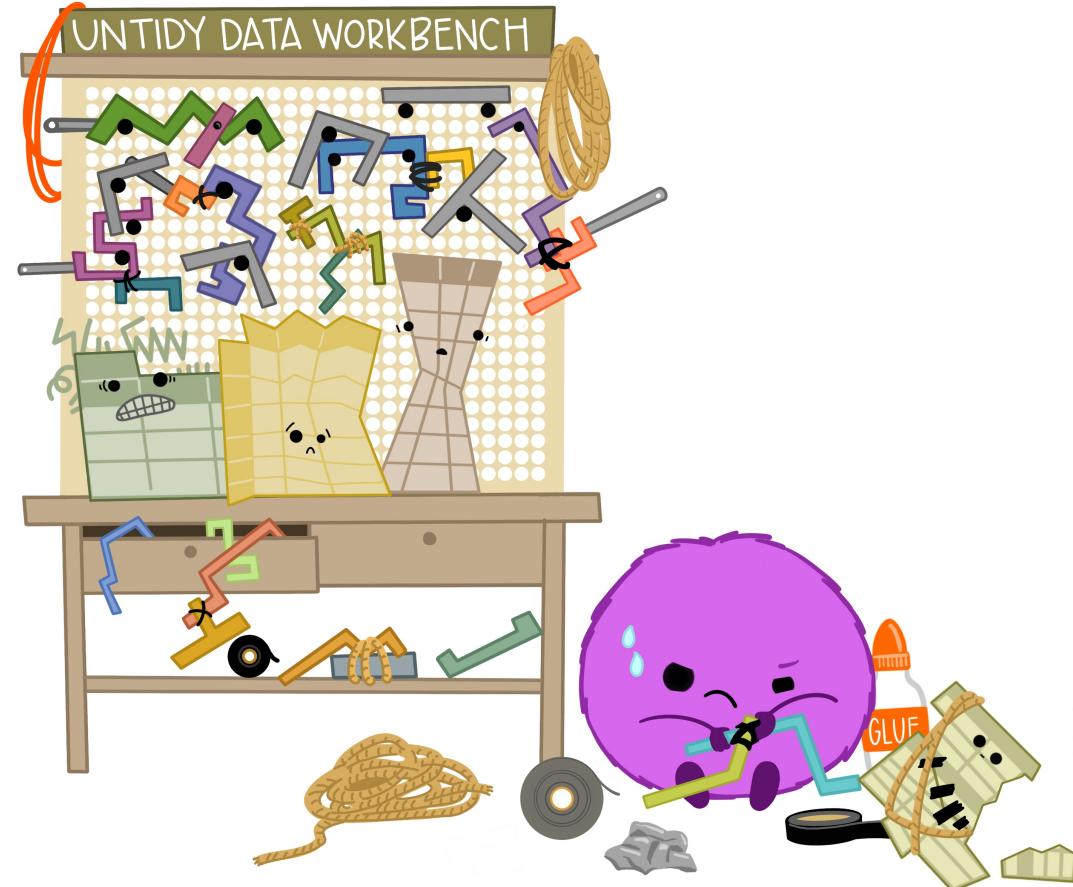


WHY "tidy" data?

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.



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 <- data.frame( # could use "tibble" instead of "data.frame" if you prefer
  time = as.Date('2009-01-01') + 0:1,
  X = rnorm(2, 0, 1),
  Y = rnorm(2, 0, 2),
  Z = rnorm(2, 0, 4)
)
stocks
```

```
##           time         X         Y         Z
## 1 2009-01-01 1.362147 -0.04054031 0.2711808
## 2 2009-01-02 1.380505 -0.65278306 3.4873908
```

We have 4 columns, the date and the stocks

How do we get this in tidy form?

1) `tidyverse::pivot_longer`

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

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

1) tidyverse::pivot_longer

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

```
## # A tibble: 6 × 3
##   time      stock    price
##   <date>    <chr>    <dbl>
## 1 2009-01-01 X        1.36
## 2 2009-01-01 Y       -0.0405
## 3 2009-01-01 Z        0.271
## 4 2009-01-02 X        1.38
## 5 2009-01-02 Y       -0.653
## 6 2009-01-02 Z        3.49
```

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

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

2) tidyverse::pivot_wider

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

```
## # A tibble: 2 × 4
##   time           X       Y     Z
##   <date>     <dbl>  <dbl> <dbl>
## 1 2009-01-01  1.36 -0.0405 0.271
## 2 2009-01-02  1.38 -0.653  3.49
```

```
tidy_stocks %>% pivot_wider(names_from = time, values_from = price)
```

```
## # A tibble: 3 × 3
##   stock `2009-01-01` `2009-01-02`
##   <chr>     <dbl>      <dbl>
## 1 X         1.36       1.38
## 2 Y        -0.0405    -0.653
## 3 Z         0.271      3.49
```

Note that the second example effectively transposed the original data

3) `tidy::separate`

```
halftime_openers <- data.frame(name = c("Dr.Dre", "Snoop.Dogg"))  
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

```
gdp <- data.frame(  
  yr = rep(2016, times = 4),  
  mnth = rep(1, times = 4),  
  dy = 1:4,  
  gdp = rnorm(4, mean = 100, sd = 2)  
)  
gdp
```

```
##      yr mnth dy      gdp  
## 1 2016     1  1 102.4538  
## 2 2016     1  2 101.3534  
## 3 2016     1  3  99.2144  
## 4 2016     1  4  95.4278
```

4) `tidyr::unite`

```
# combine "yr", "mnth", and "dy" into one "date" column
gdp %>% unite(date, c("yr", "mnth", "dy"), sep = "-")
```

```
##          date      gdp
## 1 2016-1-1 102.4538
## 2 2016-1-2 101.3534
## 3 2016-1-3  99.2144
## 4 2016-1-4  95.4278
```

4) `tidyr::unite`

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

```
gdp_u <- gdp %>% unite(date, c("yr", "mnth", "dy"), sep = "-") %>% as_tibble()  
gdp_u
```

```
## # A tibble: 4 × 2  
##   date      gdp  
##   <chr>    <dbl>  
## 1 2016-1-1 102.  
## 2 2016-1-2 101.  
## 3 2016-1-3 99.2  
## 4 2016-1-4 95.4
```

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

4) `tidyr::unite`

```
library(lubridate) # the lubridate package has helpful date conversion functions
gdp_u %>% mutate(date = ymd(date))
```

```
## # A tibble: 4 × 2
##   date      gdp
##   <date>    <dbl>
## 1 2016-01-01 102.
## 2 2016-01-02 101.
## 3 2016-01-03  99.2
## 4 2016-01-04  95.4
```

tidyr resources

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

Vignette (from the `tidyr` package)

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

Original paper (Hadley Wickham, 2014 JSS)

Relational data

What are relational data?

Multiple tables of data with pairwise relations

Relations across >2 data tables are determined by the relations between each pair

Relational data verbs from dplyr

1. **Mutating joins**: add new variables

- `left_join()`
- `right_join()`
- `inner_join()`
- `full_join()`

2. **Filtering joins**: filter observations

- `semi_join()`
- `anti_join()`

Joins

Let's learn these join commands using two small data frames

superheroes

```
## # A tibble: 7 × 4
##   name     alignment gender publisher
##   <chr>    <chr>     <chr>   <chr>
## 1 Magneto  bad       male    Marvel
## 2 Storm    good      female   Marvel
## 3 Mystique bad       female   Marvel
## 4 Batman   good      male    DC
## 5 Joker    bad       male    DC
## 6 Catwoman bad       female   DC
## 7 Hellboy  good      male    Dark Horse Comics
```

publishers

```
## # A tibble: 3 × 2
##   publisher yr Founded
##   <chr>        <int>
## 1 DC            1934
## 2 Marvel        1939
## 3 Image         1992
```

1) dplyr::left_join(x, y)

```
left_join(superheroes, publishers)

## Joining, by = "publisher"

## # A tibble: 7 × 5
##   name      alignment gender publisher      yr_founded
##   <chr>    <chr>     <chr>  <chr>          <int>
## 1 Magneto   bad       male   Marvel        1939
## 2 Storm     good      female Marvel        1939
## 3 Mystique  bad       female Marvel        1939
## 4 Batman    good      male   DC            1934
## 5 Joker     bad       male   DC            1934
## 6 Catwoman  bad       female DC            1934
## 7 Hellboy   good      male   Dark Horse Comics    NA
```

`left_join` is a **mutating join**: it adds variables to `x`

`left_join` returns all rows from `x`

2) dplyr::right_join(x, y)

```
right_join(superheroes, publishers)

## Joining, by = "publisher"

## # A tibble: 7 × 5
##   name      alignment gender publisher yr Founded
##   <chr>     <chr>    <chr>   <chr>        <int>
## 1 Magneto   bad      male    Marvel       1939
## 2 Storm     good     female  Marvel       1939
## 3 Mystique  bad      female  Marvel       1939
## 4 Batman    good     male    DC           1934
## 5 Joker     bad      male    DC           1934
## 6 Catwoman  bad      female  DC           1934
## 7 <NA>       <NA>     <NA>    Image        1992
```

right_join is a **mutating join**: it adds variables to **y**

right_join returns all rows from **y**

3) dplyr::inner_join(x, y)

```
inner_join(superheroes, publishers)

## Joining, by = "publisher"

## # A tibble: 6 × 5
##   name      alignment gender publisher yr Founded
##   <chr>    <chr>     <chr>   <chr>        <int>
## 1 Magneto  bad       male    Marvel        1939
## 2 Storm    good      female  Marvel        1939
## 3 Mystique bad       female  Marvel        1939
## 4 Batman   good      male    DC            1934
## 5 Joker    bad       male    DC            1934
## 6 Catwoman bad      female  DC            1934
```

How is `inner_join` different from `left_join` and `right_join`?

`inner_join` returns all rows in **x AND y**

4) dplyr::full_join(x, y)

```
full_join(superheroes, publishers) # how many rows will this produce?
```

```
## Joining, by = "publisher"  
  
## # A tibble: 8 × 5  
##   name     alignment gender publisher      yr_founded  
##   <chr>    <chr>     <chr>  <chr>          <int>  
## 1 Magneto  bad       male   Marvel        1939  
## 2 Storm   good      female  Marvel        1939  
## 3 Mystique bad       female  Marvel        1939  
## 4 Batman   good      male   DC            1934  
## 5 Joker    bad       male   DC            1934  
## 6 Catwoman bad       female  DC            1934  
## 7 Hellboy  good      male   Dark Horse Comics  NA  
## 8 <NA>     <NA>     <NA>   Image         1992
```

full_join returns all rows in x **OR** y

5) dplyr::semi_join(x, y)

superheroes

```
## # A tibble: 7 × 4
##   name     alignment gender publisher
##   <chr>    <chr>     <chr>  <chr>
## 1 Magneto  bad       male    Marvel
## 2 Storm    good      female  Marvel
## 3 Mystique bad       female  Marvel
## 4 Batman   good      male    DC
## 5 Joker    bad       male    DC
## 6 Catwoman bad      female  DC
## 7 Hellboy  good      male    Dark Horse Comics
```

semi_join(superheroes, publishers)

```
## Joining, by = "publisher"
## # A tibble: 6 × 4
##   name     alignment gender publisher
##   <chr>    <chr>     <chr>  <chr>
## 1 Magneto  bad       male    Marvel
## 2 Storm    good      female  Marvel
## 3 Mystique bad       female  Marvel
## 4 Batman   good      male    DC
## 5 Joker    bad       male    DC
## 6 Catwoman bad      female  DC
```

semi_join is a **filtering join**: it keeps observations in **x** that have a match in **y**

Note that the variables do not change

6) dplyr::anti_join(x, y)

superheroes

```
## # A tibble: 7 × 4
##   name    alignment gender publisher
##   <chr>    <chr>     <chr>   <chr>
## 1 Magneto  bad       male    Marvel
## 2 Storm    good      female   Marvel
## 3 Mystique bad       female   Marvel
## 4 Batman   good      male    DC
## 5 Joker    bad       male    DC
## 6 Catwoman bad       female   DC
## 7 Hellboy  good      male    Dark Horse Comics
```

anti_join(superheroes, publishers)

```
## Joining, by = "publisher"
## # A tibble: 1 × 4
##   name    alignment gender publisher
##   <chr>    <chr>     <chr>   <chr>
## 1 Hellboy  good      male    Dark Horse Comics
```

anti_join is a **filtering join**: it keeps obs. in **x** that **DO NOT** have a match in **y**

Note that the variables do not change

More on joins

Now let's use the data that comes with the **nycflights13** package

```
library(nycflights13)
flights

## # A tibble: 336,776 × 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1 2013     1     1      517            515        2       830            819
## 2 2013     1     1      533            529        4       850            830
## 3 2013     1     1      542            540        2       923            850
## 4 2013     1     1      544            545       -1      1004           1022
## 5 2013     1     1      554            600       -6      812            837
## 6 2013     1     1      554            558       -4      740            728
## 7 2013     1     1      555            600       -5      913            854
## 8 2013     1     1      557            600       -3      709            723
## 9 2013     1     1      557            600       -3      838            846
## 10 2013    1     1      558            600       -2      753            745
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
```

More on joins

planes

```
## # A tibble: 3,322 × 9
##   tailnum year type          manufacturer model engines seats speed engine
##   <chr>   <int> <chr>        <chr>      <chr>   <int> <int> <int> <chr>
## 1 N10156  2004 Fixed wing multi... EMBRAER   EMB-...     2     55    NA Turbo...
## 2 N102UW   1998 Fixed wing multi... AIRBUS INDU... A320...     2    182    NA Turbo...
## 3 N103US   1999 Fixed wing multi... AIRBUS INDU... A320...     2    182    NA Turbo...
## 4 N104UW   1999 Fixed wing multi... AIRBUS INDU... A320...     2    182    NA Turbo...
## 5 N10575   2002 Fixed wing multi... EMBRAER   EMB-...     2     55    NA Turbo...
## 6 N105UW   1999 Fixed wing multi... AIRBUS INDU... A320...     2    182    NA Turbo...
## 7 N107US   1999 Fixed wing multi... AIRBUS INDU... A320...     2    182    NA Turbo...
## 8 N108UW   1999 Fixed wing multi... AIRBUS INDU... A320...     2    182    NA Turbo...
## 9 N109UW   1999 Fixed wing multi... AIRBUS INDU... A320...     2    182    NA Turbo...
## 10 N110UW  1999 Fixed wing multi... AIRBUS INDU... A320...    2    182    NA Turbo...
## # ... with 3,312 more rows
```

More on joins

Let's perform a left join on the flights and planes data frames

- I'm going subset columns after the join, but only to keep text on the slide

```
left_join(flights, planes) %>%  
  select(year, month, day, dep_time, arr_time, carrier, flight, tailnum, type, model)
```

```
## Joining, by = c("year", "tailnum")  
  
## # A tibble: 336,776 × 10  
##   year month   day dep_time arr_time carrier flight tailnum type model  
##   <int> <int> <int>    <int>    <int> <chr>    <int> <chr>    <chr> <chr>  
## 1 2013     1     1      517      830  UA        1545 N14228  <NA> <NA>  
## 2 2013     1     1      533      850  UA        1714 N24211  <NA> <NA>  
## 3 2013     1     1      542      923  AA        1141 N619AA  <NA> <NA>  
## 4 2013     1     1      544     1004  B6        725  N804JB  <NA> <NA>  
## 5 2013     1     1      554      812  DL        461  N668DN  <NA> <NA>  
## 6 2013     1     1      554      740  UA       1696 N39463  <NA> <NA>  
## 7 2013     1     1      555      913  B6        507  N516JB  <NA> <NA>
```

More on joins

As before, `dplyr` guessed which columns to join on

It uses columns with the same name:

```
## Joining, by = c("year", "tailnum")
```

There's a problem here: the variable `year` does not have a consistent meaning across our joining datasets

In `flights` it refers to the *year of flight*, in `planes` it refers to *year of construction*

Luckily, there's an easy way to avoid this problem. How?

- Start with `?dplyr::join`

More on joins

We just need to be more explicit in the join call by using the **by =** argument

```
left_join(flights,
  planes %>% rename(year_built = year), # not necessary w/ below line, but helpful
  by = "tailnum" # be specific about the joining column
) %>%
select(year, month, day, dep_time, arr_time, carrier, flight, tailnum, year_built, type, model) %>
head(5) # just to save vertical space on the slide
```

```
## # A tibble: 5 × 11
##   year month   day dep_time arr_time carrier flight tailnum year_built type
##   <int> <int> <int>    <int>    <int> <chr>    <int> <chr>     <int> <chr>
## 1  2013     1     1      517      830  UA        1545 N14228     1999 Fixed w...
## 2  2013     1     1      533      850  UA        1714 N24211     1998 Fixed w...
## 3  2013     1     1      542      923  AA        1141 N619AA     1990 Fixed w...
## 4  2013     1     1      544     1004  B6        725  N804JB     2012 Fixed w...
## 5  2013     1     1      554      812  DL        461  N668DN     1991 Fixed w...
## # ... with 1 more variable: model <chr>
```

More on joins

What happens if we don't rename `year` before this join?

```
left_join(flights,
  planes, # not renaming "year" to "year_built" this time
  by = "tailnum") %>%
select(contains("year"), month, day, dep_time, arr_time, carrier, flight, tailnum, type, model) %>
head(5) # just to save vertical space on the slide

## # A tibble: 5 × 11
##   year.x year.y month   day dep_time arr_time carrier flight tailnum type   model
##   <int>  <int> <int> <int>    <int>    <int> <chr>   <int> <chr>   <chr> <chr>
## 1   2013    1999     1     1      517      830  UA       1545 N14228 Fixe... 737...
## 2   2013    1998     1     1      533      850  UA       1714 N24211 Fixe... 737...
## 3   2013    1990     1     1      542      923  AA       1141 N619AA Fixe... 757...
## 4   2013    2012     1     1      544     1004  B6       725  N804JB Fixe... A320...
## 5   2013    1991     1     1      554      812  DL       461  N668DN Fixe... 757...
```

What is `year.x`? What is `year.y`?

Summary

Key verbs

Import

readr

- 1. `read_csv`
- 2. `write_csv`

readxl

- 1. `read_excel`

Tidy

tidyr

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

Join

dplyr

- 1. `left_join`
- 2. `right_join`
- 3. `inner_join`
- 4. `full_join`
- 5. `semi_join`
- 6. `anti_join`

Transform

dplyr

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