Data Wrangling

Jason Willwerscheid

2022-09-12

Relational and Non-Relational Databases

A relational database stores data in a **flat** format (e.g., .csv):

pitch_type	game_date	release_speed	release_pos_x	release_pos_z	player_name	batter	pitcher	events
FF	2020-10-27	96.7	1.58	5.99	Julio Urias	642715	628711	strikeout
FF	2020-10-27	94.1	2.91	5.45	Julio Urias	642715	628711	
FF	2020-10-27	94.9	1.77	6.02	Julio Urias	642715	628711	
FF	2020-10-27	94.4	1.66	5.93	Julio Urias	670712	628711	strikeout
CU	2020-10-27	81.4	1.46	6.06	Julio Urias	670712	628711	
FF	2020-10-27	95.6	1.81	5.95	Julio Urias	670712	628711	
CU	2020-10-27	80.6	1.50	6.11	Julio Urias	670712	628711	
FF	2020-10-27	94.7	1.82	5.95	Julio Urias	670712	628711	
FF	2020-10-27	94.3	1.67	6.11	Julio Urias	670712	628711	
FF	2020-10-27	95.2	1.72	6.09	Julio Urias	622534	628711	field_out
CH	2020-10-27	87.2	1.90	5.90	Julio Urias	622534	628711	
FF	2020-10-27	94.1	1.61	6.22	Julio Urias	622534	628711	
FF	2020-10-27	94.2	1.69	6.16	Julio Urias	622534	628711	

Relational and Non-Relational Databases

- In a relational database table, the number and type of columns are **fixed**. Every new observation must include exactly those columns.
- In a non-relational data format (e.g., .json), fields can be added and subtracted at will:


```
{"ts":"2019-09-21T17:35:42Z","username":"willwerscheid","platform":"OS X 10.14.6 [x86
8]","ms_played":258463,"conn_country":"US","ip_addr_decrypted":"69.59.126.18","user_agent_decrypted":"unknown","master_meta
data_track_name":"Things It Would Have Been Helpful to Know Before the
Revolution","master_metadata_album_artist_name":"Father John Misty","master_metadata_album_album_name":"Pure
Comedy","spotify_track_uri":"spotify:track:0cLyScyKTvOEcF64ReuwiO","episode_name":null,"episode_show_name":null,"spotify_ep
isode_uri":null,"reason_start":"trackdone","reason_end":"trackdone","shuffle":false,"skipped":null,"offline":false,"offline
_timestamp":1569087083550,"incognito_mode":false},

{"ts":"2018-03-21T01:02:52Z","username":"willwerscheid","platform":"Android OS 7.0 API 24 (samsung, SM-
G930V)","ms_played":298120,"conn_country":"US","ip_addr_decrypted":"66.112.160.98","user_agent_decrypted":"unknown","master
_metadata_track_name":"The Way You Make Me Feel - 2012 Remaster","master_metadata_album_artist_name":"Michael
Jackson","master_metadata_album_album_name":"Bad 25th
Anniversary","spotify_track_uri":"spotify:track:0sKlV58cODrjxGFOyf9IXY","episode_name":null,"episode_show_name":null,"spoti
fy_episode_uri":null,"reason_start":"trackdone","reason_end":"trackdone","shuffle":false,"skipped":null,"offline":false,"of
fline_timestamp":1521593873169,"incognito_mode":false},

{"ts":"2019-08-22T17:36:34Z","username":"willwerscheid","platform":"OS X 10.14.6 [x86
8]","ms_played":238306,"conn_country":"US","ip_addr_decrypted":"69.59.126.18","user_agent_decrypted":"unknown","master_meta
data_track_name":"In the Same Room","master_metadata_album_artist_name":"Julia
Holter","master_metadata_album_album_name":"Ekstasis","spotify_track_uri":"spotify:track:63JoTd14yg1VPnpmriFVg2","episode_n
ame":null,"episode_show_name":null,"spotify_episode_uri":null,"reason_start":"trackdone","reason_end":"trackdone","shuffle"
:false,"skipped":null,"offline":false,"offline_timestamp":1566495155141,"incognito_mode":false},

{"ts":"2021-04-21T19:38:36Z","username":"willwerscheid","platform":"iOS 14.4.2 |
(iPhone13,4)","ms_played":169805,"conn_country":"US","ip_addr_decrypted":"174.194.21.106","user_agent_decrypted":"unknown",
"master_metadata_track_name":"Bones","master_metadata_album_artist_name":"Low
Roar","master_metadata_album_album_name":"Once In a Long, Long
While...","spotify_track_uri":"spotify:track:6KXohEBsBvXwYoZjbM0XcQ","episode_name":null,"episode_show_name":null,"spotify_
episode_uri":null,"reason_start":"trackdone","reason_end":"trackdone","shuffle":false,"skipped":null,"offline":false,"offli
ne_timestamp":1619033745500,"incognito_mode":false}
```

Relational and Non-Relational Databases

Non-relational databases are best for large and evolving databases, but to use them in R we need to coerce them into a flat format (and sometimes this is easier said than done):

master_metadata_track_name	master_metadata_album_artist_name	master_metadata_album_album_name
Things It Would Have Been Helpful to Know Before the...	Father John Misty	Pure Comedy
The Way You Make Me Feel - 2012 Remaster	Michael Jackson	Bad 25th Anniversary
In the Same Room	Julia Holter	Ekstasis
Bones	Low Roar	Once In a Long, Long While...
Two Promises	Sunny Day Real Estate	How It Feels To Be Something On
In A Good Way	Faye Webster	I Know I'm Funny haha
How It Feels To Be Something On	Sunny Day Real Estate	How It Feels To Be Something On
I Can't Explain - Mono Version	The Who	My Generation
Harvest Moon	Neil Young	Greatest Hits
Let's Dance - 2018 Remaster	David Bowie	Let's Dance
Half Light I	Arcade Fire	The Suburbs (Deluxe)
FANTASYNTH	Actress	AZD

Relational and Non-Relational Databases

In R, this flat structure is called a **data frame** (or, in the Tidyverse — which we will get to shortly —, a **tibble**).

Data Wrangling

Raw data is often messy. Data wrangling (or data manipulation) is the process of making data frames more suitable for the analysis that we want to do.

Data Wrangling

- In R, we will use the `dplyr` package for data wrangling.
- `dplyr` is part of the Tidyverse, which is a very popular suite of R packages designed for doing modern data science.
- You can install the entire Tidyverse by running `install.packages("tidyverse")`.

Data Wrangling

Hadley Wickham, one of the main proponents of the Tidyverse, has written about the **five verbs** of data wrangling.

Five Verbs

Problem: there are useless columns in our dataset (`spin_rate_deprecated`???).

Solution: `select()` the columns we care about.

```
select(statcast, player_name, pitch_type,
release_speed, description):
```

player_name	pitch_type	release_speed	description
Julio Urias	FF	96.7	called_strike
Julio Urias	FF	94.1	called_strike
Julio Urias	FF	94.9	swinging_strike
Julio Urias	FF	94.4	called_strike
Julio Urias	CU	81.4	ball
Julio Urias	FF	95.6	ball
Julio Urias	CU	80.6	foul

Five Verbs

Problem: we want to focus our analysis on a subset of observations.

Solution: `filter()` down to the observations we want.

```
filter(statcast, player_name == "Jacob deGrom")
```

player_name	pitch_type	release_speed	description
Jacob deGrom	SL	92.8	swinging_strike
Jacob deGrom	FF	99.3	foul
Jacob deGrom	CH	92.2	foul
Jacob deGrom	CH	89.8	ball
Jacob deGrom	CH	91.8	foul
Jacob deGrom	SL	89.8	foul

Five Verbs

Problem: the data is not sorted, or is sorted differently from how we want.

Solution: `arrange()` the data.

```
arrange(statcast, -release_speed)
```

player_name	pitch_type	release_speed	description
Jacob deGrom	FF	102.2	ball
Jacob deGrom	FF	101.1	hit_into_play
Jacob deGrom	FF	101.1	foul
Jacob deGrom	FF	101.1	ball
Jacob deGrom	FF	100.9	swinging_strike
Jacob deGrom	FF	100.8	ball
Jacob deGrom	FF	100.7	foul
Jacob deGrom	FF	100.6	foul

Five Verbs

Problem: there is an issue with a variable (e.g., it was interpreted as a string rather than a factor), or we want to create a new variable from existing variables.

Solution: `mutate()` the existing variables.

```
mutate(statcast, pitch_type = as.factor(pitch_type),
release_speed_kph = release_speed * 1.609)
```

player_name	pitch_type	release_speed	description	release_speed_kph
Jacob deGrom	FF	102.2	ball	164.4398
Jacob deGrom	FF	101.1	hit_into_play	162.6699
Jacob deGrom	FF	101.1	foul	162.6699
Jacob deGrom	FF	101.1	ball	162.6699
Jacob deGrom	FF	100.9	swinging_strike	162.3481
Jacob deGrom	FF	100.8	ball	162.1872
Jacob deGrom	FF	100.7	foul	162.0263
Jacob deGrom	FF	100.6	foul	161.8654
Jacob deGrom	FF	100.6	foul	161.8654

Five Verbs

Problem: there is too much data! We want summary statistics instead.

Solution: `summarize()` the data using a function like `mean()`, `median()`, `sum()`, or `n()` (the number of observations).

```
summarize(statcast, num_pitches = n(), median_kph =
median(release_speed_kph))
```

	num_pitches	median_kph
1	1135	150.7633

Five Verbs (and a sixth)

We can also summarize by groups using the function `group_by()`.

```
summarize(group_by(statcast, pitch_type), num_pitches
= n(), median_kph = median(release_speed_kph))
```

	pitch_type	num_pitches	median_kph
1	CH	192	147.2235
2	CU	30	135.2364
3	FF	510	158.6474
4	SL	403	148.9934

The pipe operator

If we wanted to do all these operations at once, we could write something like this:

```
summarize(
  group_by(
    mutate(
      filter(
        select(statcast,
               player_name, pitch_type, release_speed),
        player_name == "Jacob deGrom"
      ),
      pitch_type = as.factor(pitch_type),
      release_speed_kph = release_speed * 1.609
    ),
    pitch_type
  ),
  num_pitches = n(), median_kph = median(release_speed_kph)
)
```

The pipe operator

This kind of code can be really hard to read (and write!). Instead we can use Tidyverse's **pipe operator** %>%, which allows you to read the operations from top to bottom and left to right instead of inside-out:

```
statcast %>%
  select(player_name, pitch_type, release_speed) %>%
  filter(player_name == "Jacob deGrom") %>%
  mutate(pitch_type = as.factor(pitch_type),
         release_speed_kph = release_speed * 1.609) %>%
  group_by(pitch_type) %>%
  summarize(num_pitches = n(),
            median_kph = median(release_speed_kph))
```

The pipe operator

Basically, the pipe operator takes everything that happens to its left and feeds it into the function on its right as its first parameter. In other words,

```
x %>% fun(y)
```

is the same as

```
fun(x, y)
```

Joins

So far, all the data wrangling we've done has been with a single table. If we want data from another table, we need to do a **join**.

Joins

- For example, let's say that we want to know the batter's name in our Statcast dataset. We currently only have batter ID:

pitch_type	game_date	release_speed	release_pos_x	release_pos_z	player_name	batter	pitcher	events
FF	2020-10-27	96.7	1.58	5.99	Julio Urias	642715	628711	strikeout
FF	2020-10-27	94.1	2.91	5.45	Julio Urias	642715	628711	
FF	2020-10-27	94.9	1.77	6.02	Julio Urias	642715	628711	
FF	2020-10-27	94.4	1.66	5.93	Julio Urias	670712	628711	strikeout
CU	2020-10-27	81.4	1.46	6.06	Julio Urias	670712	628711	
FF	2020-10-27	95.6	1.81	5.95	Julio Urias	670712	628711	
CU	2020-10-27	80.6	1.50	6.11	Julio Urias	670712	628711	
FF	2020-10-27	94.7	1.82	5.95	Julio Urias	670712	628711	
FF	2020-10-27	94.3	1.67	6.11	Julio Urias	670712	628711	
FF	2020-10-27	95.2	1.72	6.09	Julio Urias	622534	628711	field_out
CH	2020-10-27	87.2	1.90	5.90	Julio Urias	622534	628711	
FF	2020-10-27	94.1	1.61	6.22	Julio Urias	622534	628711	
FF	2020-10-27	94.2	1.69	6.16	Julio Urias	622534	628711	

Joins

- Another dataset (provided by the Chadwick Baseball Bureau) tells us which IDs correspond to which names:

key_mlbam	name_last	name_first	name_given	name_suffix	name_matrilineal	name_nick
430911	Aardsma	David	David Allan			
NA	Aarhus	Amee	Amee			
NA	Aarns	Andrew	Andrew			
NA	Aaron	Ed	Edward			
458209	Aaron	Ging	Oginga D.			
110001	Aaron	Hank	Henry Louis			Hammer; Hammerin' Hank;
554450	Aaron	Lary	Lawrence			
554451	Aaron	Melvin	Melvin Carle			
NA	Aaron	Noah	Noah			
NA	Aaron	Robert	Robert			
NA	Aaron	Tom	Tom			
110002	Aaron	Tommie	Tommie Lee			
NA	Aaron	W. M.	W. M.			
NA	Aaron	Wil	Wilmer Calvert			
NA	Aaron	Willard	Willard Jack			
598893	Aarons	Eric	Eric			

Joins

- A **join** combines these two datasets.
- For every row in the Statcast dataset, we look at the `batter` column to get the batter ID. We then find the matching value in the `key_mlbam` column from the Chadwick dataset and add the rest of the columns from the Chadwick dataset to that observation:

pitch_type	game_date	release_speed	release_pos_x	release_pos_z	player_name	batter	name_last	name_first	name_given
FF	2020-10-27	96.7	1.58	5.99	Julio Urias	642715	Adames	Willy	Willy Rafael
FF	2020-10-27	94.1	2.91	5.45	Julio Urias	642715	Adames	Willy	Willy Rafael
FF	2020-10-27	94.9	1.77	6.02	Julio Urias	642715	Adames	Willy	Willy Rafael
FF	2020-10-27	94.4	1.66	5.93	Julio Urias	670712	Brosseau	Mike	Michael Dillon
CU	2020-10-27	81.4	1.46	6.06	Julio Urias	670712	Brosseau	Mike	Michael Dillon
FF	2020-10-27	95.6	1.81	5.95	Julio Urias	670712	Brosseau	Mike	Michael Dillon
CU	2020-10-27	80.6	1.50	6.11	Julio Urias	670712	Brosseau	Mike	Michael Dillon
FF	2020-10-27	94.7	1.82	5.95	Julio Urias	670712	Brosseau	Mike	Michael Dillon
FF	2020-10-27	94.3	1.67	6.11	Julio Urias	670712	Brosseau	Mike	Michael Dillon
FF	2020-10-27	95.2	1.72	6.09	Julio Urias	622534	Margot	Manuel	Manuel
CH	2020-10-27	87.2	1.90	5.90	Julio Urias	622534	Margot	Manuel	Manuel
FF	2020-10-27	94.1	1.61	6.22	Julio Urias	622534	Margot	Manuel	Manuel

Joins

This example is straightforward because:

1. The Chadwick dataset includes every MLB player.
2. The IDs in the Chadwick dataset are **unique** (that is, when we use the ID to look up a row, we get exactly one player).

Types of Joins

One of the most common types of joins is a **left join**, which always returns at least one row for every row in the first table and returns multiple rows if it finds multiple matches. In our example:

1. If it didn't find the batter in the Chadwick dataset, it would return that observation with `NA` values in all of the columns added from the Chadwick dataset (the name columns).
2. If it found multiple matches in the `key_mlbam` column from the Chadwick dataset, it would return one row for each match.

Types of Joins

Another common join is an **inner join**, which only returns rows when one or more matches are found:

1. If it didn't find the batter in Chadwick, it would simply "drop" that observation from the result.
2. If it found multiple matches, it would return multiple rows (as in a left join).

Types of Joins

A **right join** is just the inverse of a left join. In other words, `left_join(A, B)` is the same as `right_join(B, A)`.

- What happens if I use a right join instead of a left join in our example?

Example

To put everything together, let's see which hitter hit the most home runs off of Jacob deGrom.

- Which columns do we need from the Statcast dataset?

Example

To put everything together, let's see which hitter hit the most home runs off of Jacob deGrom.

- Which columns do we need from the Statcast dataset?
- What other columns do we need, and how do we get them?

Example

To put everything together, let's see which hitter hit the most home runs off of Jacob deGrom.

- Which columns do we need from the Statcast dataset?
- What other columns do we need, and how do we get them?
- Which rows do we need?

Example

To put everything together, let's see which hitter hit the most home runs off of Jacob deGrom.

- Which columns do we need from the Statcast dataset?
- What other columns do we need, and how do we get them?
- Which rows do we need?
- How should we group the data?

Example

To put everything together, let's see which hitter hit the most home runs off of Jacob deGrom.

- Which columns do we need from the Statcast dataset?
- What other columns do we need, and how do we get them?
- Which rows do we need?
- How should we group the data?
- How should we summarize?

Example

To put everything together, let's see which hitter hit the most home runs off of Jacob deGrom.

- Which columns do we need from the Statcast dataset?
- What other columns do we need, and how do we get them?
- Which rows do we need?
- How should we group the data?
- How should we summarize?
- How can we put the answer at the top of the table?

Example

```
res <- statcast %>%
  rename(pitcher_name = player_name) %>%
  select(pitcher_name, events, batter) %>%
  left_join(chadwick, by = c("batter" = "key_mlbam")) %>%
  mutate(batter_name = paste(name_first, name_last)) %>%
  select(pitcher_name, batter_name, events) %>%
  filter(pitcher_name == "Jacob deGrom",
         events == "home_run") %>%
  group_by(batter_name) %>%
  summarize(num_HR = n()) %>%
  arrange(-num_HR)
```

Example

	batter_name	num_HR
1	Andrew Stevenson	2
2	Andrew Knapp	1
3	Garrett Cooper	1
4	Jesus Aguilar	1
5	Nathaniel Lowe	1
6	Travis d'Arnaud	1