Chess Tournament Database Pipeline

AUTHOR
Kevin Havis

Setup

```
library(tidyverse)
library(purrr)
library(DBI)
library(duckdb)
library(tools)
library(knitr)
```

Introduction

We'll be parsing a text file containing the results of recent chess tournament in this article. By leveraging `dplyr` and `tidyverse` libraries, we can convert this text file into a proper structured data set. Once we have the data formatted, we can review the results of the tournament.

Loading & cleaning the data

This data is easily human-readable, however it has a couple things we need to address to get it into proper shape. Let's start with cleaning up the ASCII delimiters.

```
-----------------------------------------------------------------------------
 Pair | Player Name                     |Total|Round|Round|Round|Round|Round|
 Num  | USCF ID / Rtg (Pre->Post)       | Pts |  1  |  2  |  3  |  4  |  5  |
-----------------------------------------------------------------------------
    1 | GARY HUA                        |6.0  |W  39|W  21|W  18|W  14|W   7|
   ON | 15445895 / R: 1794   ->1817     |N:2  |W    |B    |W    |B    |W    |
-----------------------------------------------------------------------------
    2 | DAKSHESH DARURI                 |6.0  |W  63|W  58|L   4|W  17|W  16|
```

```
   MI | 14598900 / R: 1553   ->1663     |N:2  |B    |W    |B    |W    |B    |
-------------------------------------------------------------------------------
    3 | ADITYA BAJAJ                     |6.0  |L   8|W  61|W  25|W  21|W  11|
   MI | 14959604 / R: 1384   ->1640     |N:2  |W    |B    |W    |B    |W    |
```

ASCII clean up

The headers, rows, and columns are delimited by ASCII characters, which we will need to strip out. We'll do this by reading each line in as a row and delimiting the columns.

```
        # Pre-define our initial column names
        init_cols = c(
          'pairNum_state',
          'player_info',
          'total',
          'round_1',
          'round_2',
          'round_3',
          'round_4',
          'round_5',
          'round_6',
          'round_7'
        )

        # Create a tibble by splitting row-wise by line breaks
        # and column-wise by pipe
        df <- tibble(txt = read_lines('tournament_info.txt')) |>
          separate_rows(sep = '\\n') |>
          separate(col = txt, into = init_cols, sep = '\\|')

        # Drop --- row separators
        df <- na.omit(df)

        # Drop secondary header
        df <- df |>
          slice(-(1:2))
```

pairNum_state	player_info	total	round_1	round_2	round_3	round_4	round_5	round_6	round_7
1	GARY HUA	6.0	W 39	W 21	W 18	W 14	W 7	D 12	D 4
ON	15445895 / R: 1794 ->1817	N:2	W	B	W	B	W	B	W
2	DAKSHESH DARURI	6.0	W 63	W 58	L 4	W 17	W 16	W 20	W 7
MI	14598900 / R: 1553 ->1663	N:2	B	W	B	W	B	W	B
3	ADITYA BAJAJ	6.0	L 8	W 61	W 25	W 21	W 11	W 13	W 12
MI	14959604 / R: 1384 ->1640	N:2	W	B	W	B	W	B	W

Ungrouping the rows

With the extra ASCII characters cleaned up, we're in much better shape. We can now turn our attention to the stacked rows for each player. To do this, we'll group the rows and then pivot them wider into their own columns.

```
# We have two rows per player so we group them and create new columns
df <- df |>
  group_by(grp = ceiling(row_number()/2)) |> # create groups by rounding the mod of each row numb
  mutate(row = row_number()) |>
  pivot_wider(names_from = row, values_from = -c(row, grp)) |> # break out row pairs into columns
  ungroup() |>
  select(-grp) |>
  mutate(across(everything(), ~ str_trim(.x)))
```

```
# Update column names
col_names = c(
  "player",
  "state",
  "name",
  "rating_info",
  "total_pts",
  "n",
  "results_rd_1",
  "color_rd_1",
```

```
    "results_rd_2",
    "color_rd_2",
    "results_rd_3",
    "color_rd_3",
    "results_rd_4",
    "color_rd_4",
    "results_rd_5",
    "color_rd_5",
    "results_rd_6",
    "color_rd_6",
    "results_rd_7",
    "color_rd_7"
  )

  df <- df |> set_names(col_names)
```

player	state	name	rating_info	total_pts	n	results_rd_1	color_rd_1	results_rd_2	color_rd_2	results_rd_3	color_rd_3
1	ON	GARY HUA	15445895 / R: 1794 ->1817	6.0	N:2	W 39	W	W 21	B	W 18	W
2	MI	DAKSHESH DARURI	14598900 / R: 1553 ->1663	6.0	N:2	W 63	B	W 58	W	L 4	B
3	MI	ADITYA BAJAJ	14959604 / R: 1384 ->1640	6.0	N:2	L 8	W	W 61	B	W 25	W
4	MI	PATRICK H SCHILLING	12616049 / R: 1716 ->1744	5.5	N:2	W 23	W	D 28	B	W 2	W
5	MI	HANSHI ZUO	14601533 / R: 1655 ->1690	5.5	N:2	W 45	B	W 37	W	D 12	B

player	state	name	rating_info	total_pts	n	results_rd_1	color_rd_1	results_rd_2	color_rd_2	results_rd_3	color_rd_3
6	OH	HANSEN SONG	15055204 / R: 1686 ->1687	5.0	N:3	W 34	W	D 29	B	L 11	W

Concatenated values

Next, we'll need to address the concatenated values, including the `rating_info` which contains the player's USCF ID, rating before the tournament, and rating after the tournament.

We'll also split the "results" columns for each round into the result (W / L) and the opponent's ID number using regex.

```
# Split results columns into single columns
df <- df |>
 separate_wider_regex(
   starts_with("results"),
   patterns = c(
     result = "[WLDBH]",
     opponent = "\\s+\\d+"),
   names_sep = "_",
   too_few = "align_start") |>
  mutate(across(starts_with("results"), ~ str_trim(.x)))
```

```
# Create interim tibble
rating_df <- df |>
  filter(str_detect(rating_info, "[0-9]{8}")) |>
  select(rating_info)

# Retrieve USCF ID
uscf_id <- rating_df |>
  mutate(uscf_id = str_extract(rating_info, "([0-9]{8})")) |>
  select(uscf_id)

# Retrieve pre_rating
```

```
pre_rating <- rating_df |>
  mutate(pre_rating = str_extract(rating_info, "(?<=R:\\s{1,5})\\d+")) |>
  select(pre_rating)

# Retrieve post_rating
post_rating <- rating_df |>
  mutate(post_rating = str_extract(rating_info, "(?<=>\\s{0,5})\\d{3,4}")) |>
  select(post_rating)

# Add new columns to main tibble
df <- df |>
  cbind(uscf_id) |>
  cbind(pre_rating) |>
  cbind(post_rating) |>
  mutate(pre_rating = as.numeric(pre_rating)) |>
  mutate(post_rating = as.numeric(post_rating)) |>
  select(everything(), -rating_info)
```

We now have a well organized and “tidy” data set!

player	state	name	total_pts	n	results_rd_1_result	results_rd_1_opponent	color_rd_1	results_rd_2_result	results_rd_2
1	ON	GARY HUA	6.0	N:2	W	39	W	W	21
2	MI	DAKSHESH DARURI	6.0	N:2	W	63	B	W	58
3	MI	ADITYA BAJAJ	6.0	N:2	L	8	W	W	61
4	MI	PATRICK H SCHILLING	5.5	N:2	W	23	W	D	28
5	MI	HANSHI ZUO	5.5	N:2	W	45	B	W	37
6	OH	HANSEN SONG	5.0	N:3	W	34	W	D	29

Save into database

While this data is useful and we could certainly perform any further analysis we'd like with it, we should store it properly in a database. This will allow us manage the data in a scalable way to accommodate new players and future tournaments.

Prepare our tables

We'll first split the data into logical tables, `players` and `rounds`. The `players` data is already well organized, but we'll add some finishing touches.

```
# Separate data into player table

players <- df |>
  select(player, state, name, pre_rating, post_rating, total_pts) |>
  mutate(rating_change = post_rating - pre_rating) |>
  mutate(name = str_to_title(name))
```

The `rounds` table will contain information specific to this tournament. To make this more database-friendly, we'll convert the data into a long format.

```
# Convert round columns into long format
rounds <- df |>
  pivot_longer(
    cols = starts_with("results_rd_"),
    names_to = c("round", "type"),
    names_pattern = "results_rd_(\\d+)_(result|opponent)",
    values_to = "value"
  ) |>
  pivot_longer(
    cols = starts_with("color"),
    names_to = "color_col",
    values_to = "color"
  ) |>
  pivot_wider(
    names_from = c("type"),
```

```
    values_from = c("value")
  ) |>
  select(player, color, round, result, opponent)
```

Write to the database

Now we can actually add our new tables to our database.

```
# Give rounds a unique ID as a key
rounds <- rounds |>
  mutate(id = row_number())

conn <- DBI::dbConnect(duckdb::duckdb(), dbdir = "tournament.db")

# Write tables if they don't already exist
if (!dbExistsTable(conn, "players")) {
dbWriteTable(conn, "players", players)
} else {

}


if (!dbExistsTable(conn, "rounds")) {
dbWriteTable(conn, "rounds", rounds)
} else {

}


dbDisconnect(conn)
```

Tournament results

Now, finally, we can calculate the final results of the tournament.

We'll calculate the average rating of each player's opponents.

```
# Calculate average opponent's rating

average_ratings <- left_join(rounds, players, by = c("opponent" = "player")) |>
  group_by(player) |>
  summarize(average_opponent_rating = mean(pre_rating, na.rm = TRUE))

players <- left_join(players, average_ratings, by = "player")
```

Build our final table

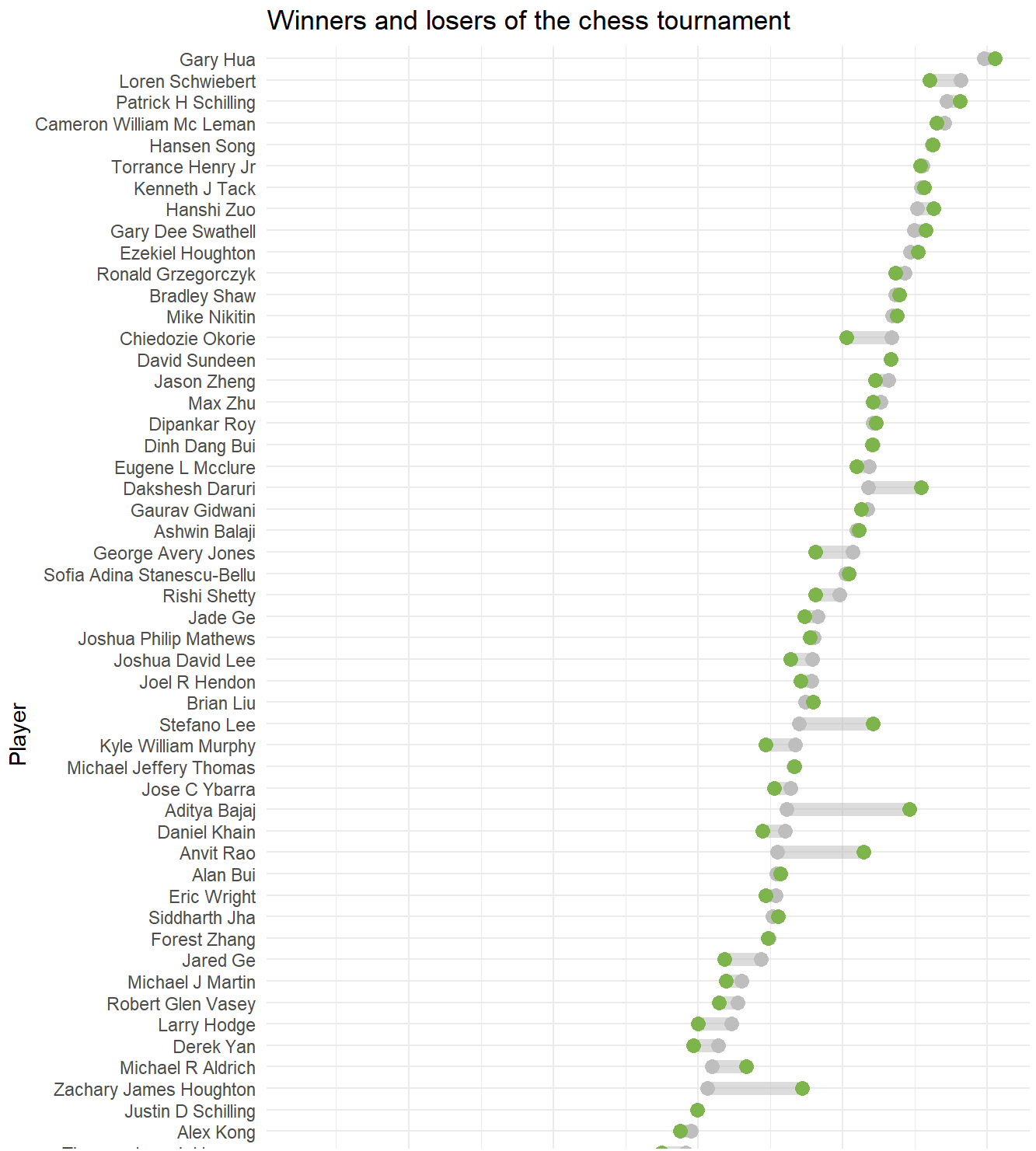
Our final table contains the player number, the player's name, their home state, the total points they won through the tournament, their rating before the tournament, and finally, our newly calculated average opponent's rating.

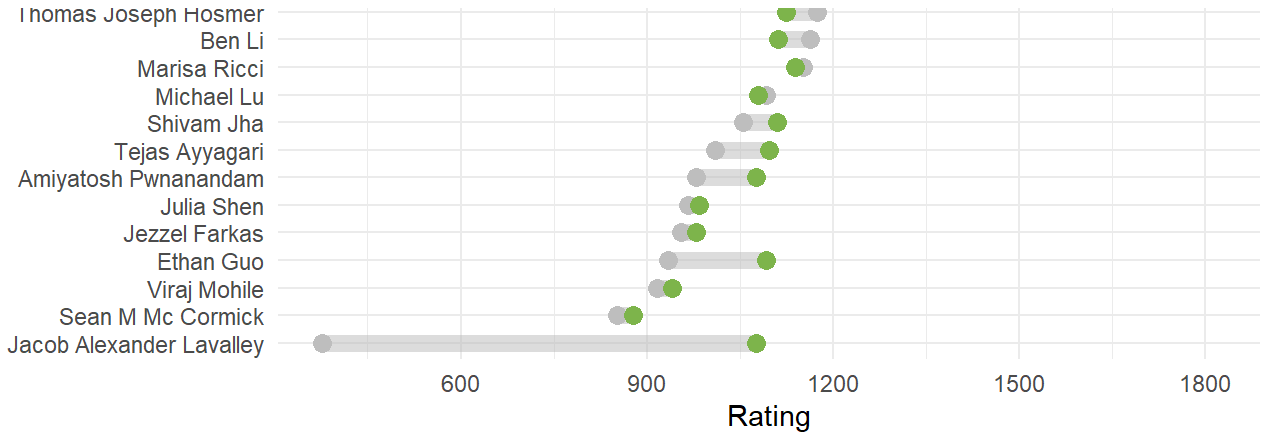
```
# Create final table

final_table <- players |>
  select(player, name, state, total_pts, pre_rating, average_opponent_rating)
```

player	name	state	total_pts	pre_rating	average_opponent_rating
1	Gary Hua	ON	6.0	1794	1605.286
2	Dakshesh Daruri	MI	6.0	1553	1469.286
3	Aditya Bajaj	MI	6.0	1384	1563.571
4	Patrick H Schilling	MI	5.5	1716	1573.571
5	Hanshi Zuo	MI	5.5	1655	1500.857
6	Hansen Song	OH	5.0	1686	1518.714

Winners and losers

Now that we have all our data in order, we can review the results of the tournament. Let's visualize the change in ratings after the tournament.


Winners and losers of the chess tournament
Player
Gary Hua
Loren Schwiebert
Patrick H Schilling
Cameron William Mc Leman
Hansen Song
Torrance Henry Jr
Kenneth J Tack
Hanshi Zuo
Gary Dee Swathell
Ezekiel Houghton
Ronald Grzegorczyk
Bradley Shaw
Mike Nikitin
Chiedozie Okorie
David Sundeen
Jason Zheng
Max Zhu
Dipankar Roy
Dinh Dang Bui
Eugene L Mcclure
Dakshesh Daruri
Gaurav Gidwani
Ashwin Balaji
George Avery Jones
Sofia Adina Stanescu-Bellu
Rishi Shetty
Jade Ge
Joshua Philip Mathews
Joshua David Lee
Joel R Hendon
Brian Liu
Stefano Lee
Kyle William Murphy
Michael Jeffery Thomas
Jose C Ybarra
Aditya Bajaj
Daniel Khain
Anvit Rao
Alan Bui
Eric Wright
Siddharth Jha
Forest Zhang
Jared Ge
Michael J Martin
Robert Glen Vasey
Larry Hodge
Derek Yan
Michael R Aldrich
Zachary James Houghton
Justin D Schilling
Alex Kong



Conclusion

In this article we developed a full end-to-end pipeline to parse, organize, store, and visualize the results of a chess tournament. We used common packages from the `tidyverse` and leaned heavily on regex to extract the key values out of our data. Once the data was in tidy shape, we separated it into logical relational tables and uploaded to our database. Finally, we plotted the outcomes of the tournament to identify the winners and losers of the event.