Lichess Analysis Progress Check

In Journal Form

Andrew Zhang

1. Scraping with Lichess API

i. Getting a Username List - Methodology

I want to emulate the data within Lichess datasets available on Kaggle, which means accessing the game data of a bunch of Lichess users.

To access the data, I first had to learn how to connect to the Lichess API with Python so that it could provide me the data I needed. They have official clients like Berserk (they seem to act as libraries you can import to make it easier to scrape data) but I wasn't quite sure how to make that work. I eventually figured out that you could generate your own API key to manually request data from Lichess instead of using a middle-man client.

I first started with exploring the API a bit. There is a way to get specific games from a specific user, but you also need to input the username into the URL that you request from for that to happen, meaning that I would need a list of usernames. I looked through the API documentation and eventually understood the reasoning behind procedures of previous scrapers of Lichess game data: logging popular team rosters. Lichess has its own URL that you can query from with the input of a team name that returns a list of all members in a team. Thus, if you query a bunch of popular teams, you could get a bunch of player usernames.

And thus the general algorithm arises: Loop through a list of teams (that I would probably manually input through filtering high-population teams on their website), querying the full team roster for each team. Then, loop through all usernames gathered and scrape data from each of their games (I would just do 10 latest games as the volume of data was getting a bit much).

ii. Getting a Username List - Application

Overall my beginning code was structured, and will likely remain structured, like so:

- Imports and global definitions (I save my API key here)
- User data function - requests game data of a specific user
- Processing function - unpack the data received into a dictionary format, each key being a column that will be in the CSV
    - For rudimentary code testing purposes, I'm using the basic categories for now:
        - GameID
        - Rated (bool)
        - Start time
        - End time
        - Turns
        - Status
        - Winner
        - Increment (the time control)
        - white_id (player ID on the white side)
        - white_rating
        - black_rating
        - black_id
        - Moves
        - Opening_code
        - Opening_name
        - Opening_ply
    - When I know that my code works, I will then probably filter out the games more so that I can only have the ones that have been analyzed by a Chess engine, which will then allow me to get performance (# of mistakes of different categories)
- Saving function - pretty simple function, just turns the dictionary into a Pandas dataframe and then turns the data frame into a csv, which does the rest of the work in formatting for you

Starting off, the API was quite…finicky. After generating my API token and following the instructions on the official Lichess API documentation, I was able to successfully access my own account's information. However, whenever I tried to access specific team information, it always returned Error 403, which corresponds to a lack of authorization. This didn't really make sense seeing how the documentation itself states that as long as a team is public, calling the API for that specific team with an API token would allow me to access it. Yet, despite trying it on the "coders" team (a team Lichess specifically made for testing), it still returned 403. For some time, I assumed that Lichess had recently made a change to up their data security, and kept researching to find if I could get an API token that had higher access.

The eventual fix I found? Just not using the API key. Removing the API key as an argument when requesting for team data resulted in the data being returned. The rest of debugging involved using the correct syntax when parsing the "JSON" returned from querying data. After a bunch more documentation reading, I figured out that data was being returned in a PGN (Portable Game Notation), and also that the requests library is able to handle that type of data, which is pretty cool.

After I finally sorted out the formatting issues, it started to actually take data from each player within each team. Then a new problem arose:

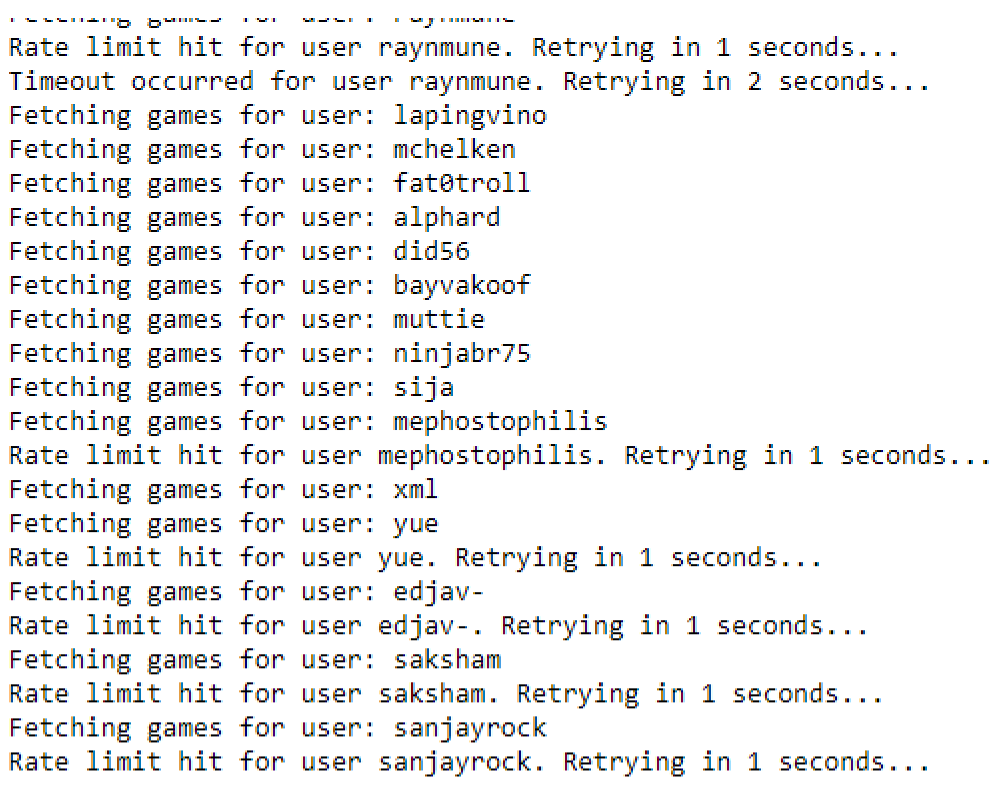
```
ConnectionError: HTTPSConnectionPool(host='lichess.org', port=443): Max retries
exceeded with url: /api/games/user/chessking900?max=10&format=pgn
```

```
TimeoutError: [WinError 10060] A connection attempt failed because the connected
party did not properly respond after a period of time, or established connection
failed because connected host has failed to respond
```

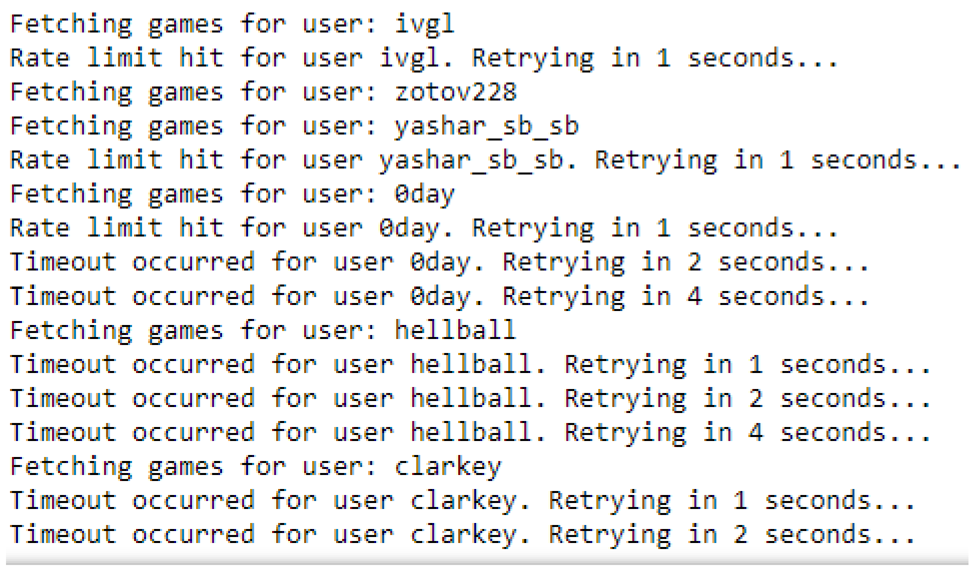
Wooo, networking issues! The first error indicates that I was hitting the Lichess API "rate limit," a limit set for the amount of requests you can send during a certain time frame. A pretty obvious answer to this is to slow down requests (by adding a delay). Apparently another fix is to use an API token….but I've seen how that goes.

The second error is a direct result of the first error as it is a timeout error, meaning that the code tried too long to connect to the Lichess server without any response and as a result decided to terminate itself. I.e., Lichess got upset that I was requesting too much data and stopped responding, then the code saw that Lichess stopped responding, decided there was no point in continuing to try, and stopped.

I added some time.sleep()s around the user game collection code and also added some branching print statements to check for when either of these errors occurred. THIS final draft allowed me to finally save a csv file of all of the data.



```
Rate limit hit for user raynmune. Retrying in 1 seconds...
Timeout occurred for user raynmune. Retrying in 2 seconds...
Fetching games for user: lapingvino
Fetching games for user: mchelken
Fetching games for user: fat0troll
Fetching games for user: alphard
Fetching games for user: did56
Fetching games for user: bayvakoof
Fetching games for user: muttie
Fetching games for user: ninjabr75
Fetching games for user: sija
Fetching games for user: mephostophilis
Rate limit hit for user mephostophilis. Retrying in 1 seconds...
Fetching games for user: xml
Fetching games for user: yue
Rate limit hit for user yue. Retrying in 1 seconds...
Fetching games for user: edjav-
Rate limit hit for user edjav-. Retrying in 1 seconds...
Fetching games for user: saksham
Rate limit hit for user saksham. Retrying in 1 seconds...
Fetching games for user: sanjayrock
Rate limit hit for user sanjayrock. Retrying in 1 seconds...
```



```
Fetching games for user: ivgl
Rate limit hit for user ivgl. Retrying in 1 seconds...
Fetching games for user: zotov228
Fetching games for user: yashar_sb_sb
Rate limit hit for user yashar_sb_sb. Retrying in 1 seconds...
Fetching games for user: 0day
Rate limit hit for user 0day. Retrying in 1 seconds...
Timeout occurred for user 0day. Retrying in 2 seconds...
Timeout occurred for user 0day. Retrying in 4 seconds...
Fetching games for user: hellball
Timeout occurred for user hellball. Retrying in 1 seconds...
Timeout occurred for user hellball. Retrying in 2 seconds...
Timeout occurred for user hellball. Retrying in 4 seconds...
Fetching games for user: clarkey
Timeout occurred for user clarkey. Retrying in 1 seconds...
Timeout occurred for user clarkey. Retrying in 2 seconds...
```

I wanted to try to access the member list to see what point the code was on but I think the code was taking so much of the resources my specific connection was allotted that I wasn't even able to access the site manually. I just had to wait (a LONG time) for this code to finish scraping the games of all users.

Something interesting to note about this data scraping is that the timeouts and rate limits were not consistent. You'd think that once I hit a limit, it would consistently rate-limit error between successes, yet I had bursts of successes and then some oscillating errors, then more bursts of successes. My guess is that the ones my code errored on were the more active players, and thus more people were requesting data from those players from checking their profile url.

iii. Data Format

The data—theoretically—comes in this column format:

- Event
- Site
- date
- white
- black
- result
- white_elo
- Black_elo
- Time_control
- Eco
    - This is the ECO Opening code (a code associated with what opening was played)
- Termination (surrender or checkmate)
- Moves

In reality (my parsing code was, once again, subpar), each row comes in groups of two, where the first row has the termination type repeat twice and the second row has all values as "Unknown" except the Moves column where it actually shows the entire game's notation:

Row 1:

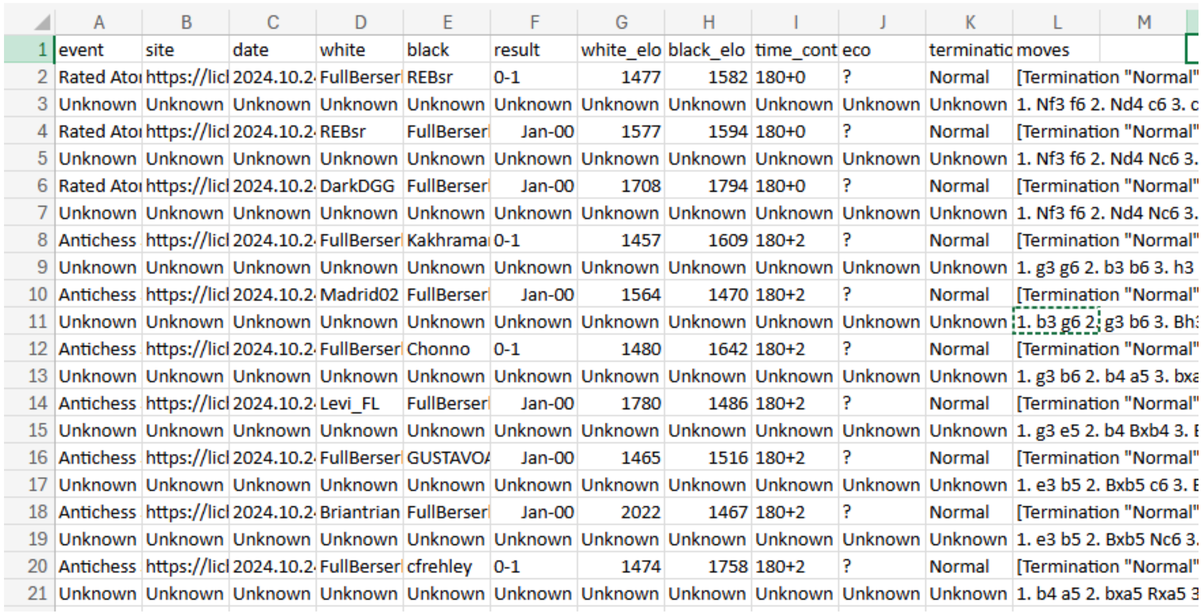
- Rated Atomic game
- https://lichess.org/TW9PFYX7
- 2024.10.24
- FullBerserker
- REBsr
- 0-1
- 1477
- 1582
- 180+0
- ?
- Normal

- [Termination "Normal"]

Row 2:

- Unknown
- Unknown
- Unknown
- Unknown
- Unknown
- Unknown
- Unknown
- Unknown
- Unknown
- Unknown
- Unknown
- 1. Nf3 f6 2. Nd4 c6 3. c4 e6 4. Qb3 Bb4 5. Nc3 Bxc3 6. d4 Qa5+ 7. b4 Qa4 8. f3 Qd1+ 9. Kf2 Qxe2# 0-1

More clearly (there's about 6000 rows of this):



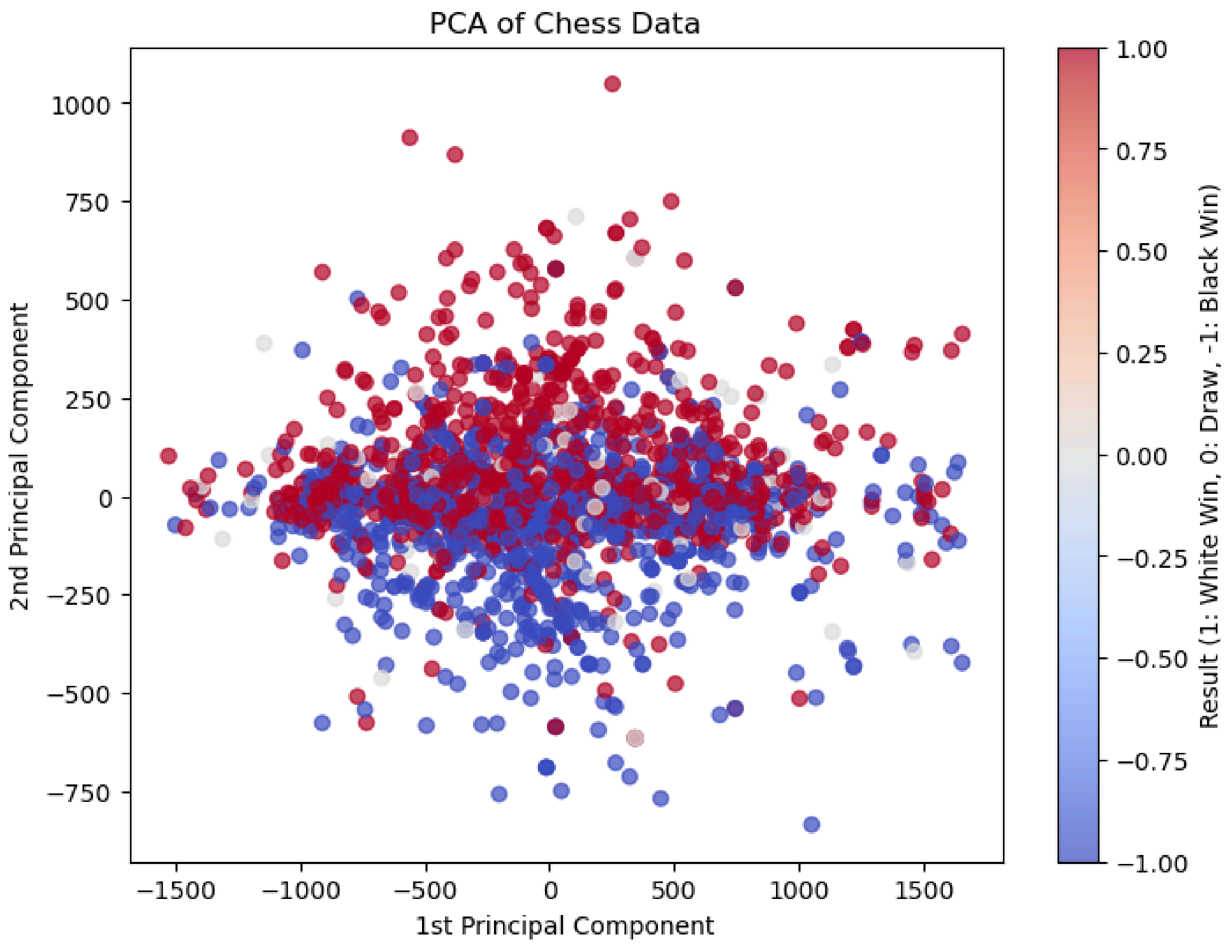
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	event	site	date	white	black	result	white_elo	black_elo	time_cont	eco	terminatic	moves	
2	Rated Ato	https://lic	2024.10.2	FullBerser	REBsr	0-1	1477	1582	180+0	?	Normal	[Termination "Normal"	
3	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	1. Nf3 f6 2. Nd4 c6 3. c	
4	Rated Ato	https://lic	2024.10.2	REBsr	FullBerser	Jan-00	1577	1594	180+0	?	Normal	[Termination "Normal"	
5	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	1. Nf3 f6 2. Nd4 Nc6 3.	
6	Rated Ato	https://lic	2024.10.2	DarkDGG	FullBerser	Jan-00	1708	1794	180+0	?	Normal	[Termination "Normal"	
7	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	1. Nf3 f6 2. Nd4 Nc6 3.	
8	Antichess	https://lic	2024.10.2	FullBerser	Kakhrama	0-1	1457	1609	180+2	?	Normal	[Termination "Normal"	
9	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	1. g3 g6 2. b3 b6 3. h3	
10	Antichess	https://lic	2024.10.2	Madrid02	FullBerser	Jan-00	1564	1470	180+2	?	Normal	[Termination "Normal"	
11	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	1. b3 g6 2. g3 b6 3. Bh	
12	Antichess	https://lic	2024.10.2	FullBerser	Chonno	0-1	1480	1642	180+2	?	Normal	[Termination "Normal"	
13	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	1. g3 b6 2. b4 a5 3. bxa	
14	Antichess	https://lic	2024.10.2	Levi_FL	FullBerser	Jan-00	1780	1486	180+2	?	Normal	[Termination "Normal"	
15	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	1. g3 e5 2. b4 Bxb4 3.	
16	Antichess	https://lic	2024.10.2	FullBerser	GUSTAVO	Jan-00	1465	1516	180+2	?	Normal	[Termination "Normal"	
17	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	1. e3 b5 2. Bxb5 c6 3.	
18	Antichess	https://lic	2024.10.2	Briantrian	FullBerser	Jan-00	2022	1467	180+2	?	Normal	[Termination "Normal"	
19	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	1. e3 b5 2. Bxb5 Nc6 3.	
20	Antichess	https://lic	2024.10.2	FullBerser	cfrehley	0-1	1474	1758	180+2	?	Normal	[Termination "Normal"	
21	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	1. b4 a5 2. bxa5 Rxa5 3	

This isn't too big of a deal; I'll just edit the data from the analysis code for now, but in the future it can be fixed in the parsing stage. Onto the analysis!

2. 3D Analysis

i. PCA Processing

I wanted to start with just the data that is, by default, numeric, and thus I began with a dimension size of 3: white elo, black elo, result_encoded (1 for white win, 0 for a draw, -1 for black win). This allows us to explore a very commonly analyzed relationship between Elo rating (skill level) and game result, and also more easily glean relationships due to the few variables.

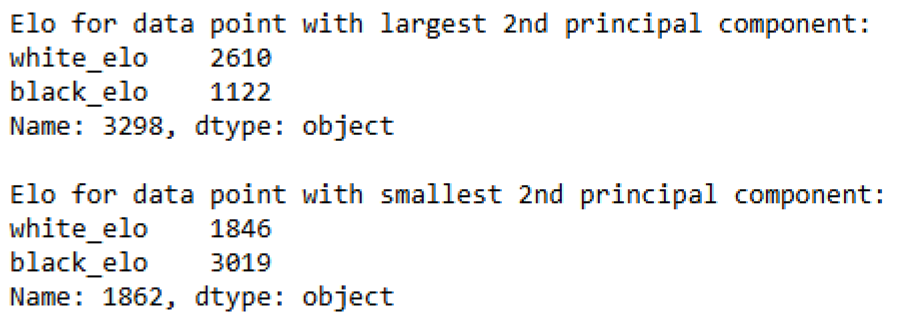


Cool! It seems that many of the games bunch up in the center, black wins take over on the bottom half of the graph, and white wins take over the upper half. This is not by design, and thus it suggests a pattern in the data.

Just thinking about skill level's influence on winning a game, I think it is likely that the central areas, where red and blue are mixed, likely represent games where the white and black elo ratings are relatively close, leading to a more balanced outcome distribution (this would also come with the indication that evenly matched Elo players make game results less predictable).

As a result, blue gathering on the bottom and red gathering on the top likely suggest games where the black/white player's Elo rating was higher by some significant margin compared to their opponent. Since we only have ELO ratings and game results as our variables, these patterns would then suggest that Elo disparity influences game outcomes (perhaps a higher disparity = more predictable outcome).

I'll run a quick sanity check to see if the data follows my assumptions:

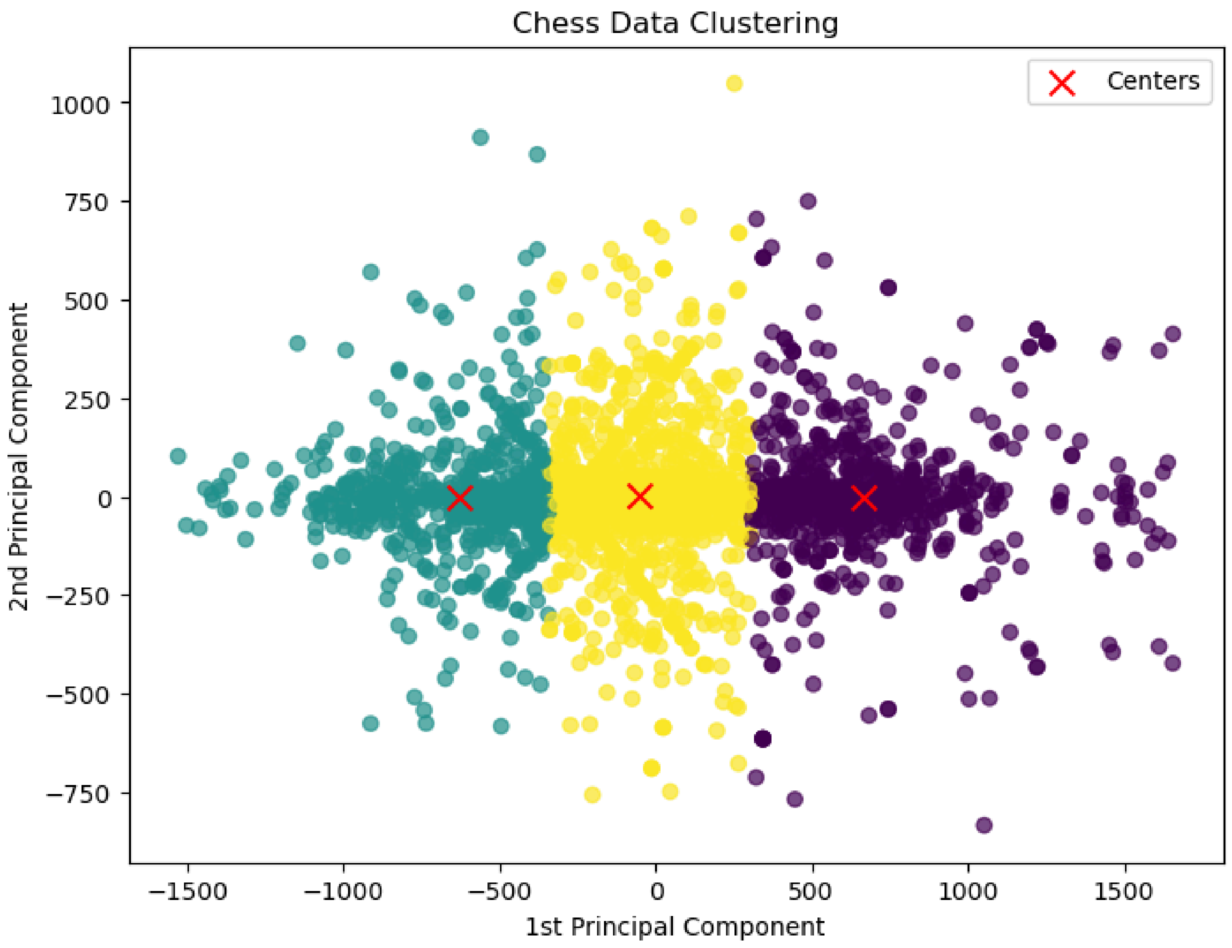


```
Elo for data point with largest 2nd principal component:
white_elo    2610
black_elo    1122
Name: 3298, dtype: object

Elo for data point with smallest 2nd principal component:
white_elo    1846
black_elo    3019
Name: 1862, dtype: object
```

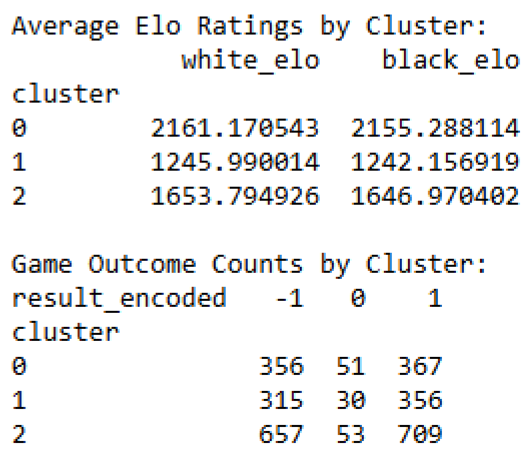
Bingo! *I can't imagine how much the lower Elo players were freaking out when they got matched for these games…*

ii. KMeans Clustering



This result is… a bit vexing. In contrast to the PCA graph, which seemed to be divided horizontally, the clusters here are divided vertically. However, upon closer consideration, I don't think this is too surprising.

The PCA graph indicated that Elo differences influenced game results. On the other hand, the KMeans algorithm tries to find the best clusters based on data point distance. It is clear from the graph that more variance is captured when looking up vertically from the x-axis (points on one x value are spread out a lot more than points on one y value), so I don't think it's that surprising that the dividing lines are oriented vertically. To match our assumptions/gleaned ideas from the PCA graph, I examined the elo rating averages within each cluster and then the game results by cluster. I expected the wins from each side to be close for each cluster and the average Elo to be about the same as well since each cluster seems to be symmetric across the x axis.



```
Average Elo Ratings by Cluster:
           white_elo    black_elo
cluster
0        2161.170543  2155.288114
1        1245.990014  1242.156919
2        1653.794926  1646.970402

Game Outcome Counts by Cluster:
result_encoded   -1   0    1
cluster
0               356  51  367
1               315  30  356
2               657  53  709
```

Whew! I was right! Thus, it seems that KMeans clustering grouped games not by Elo disparity but by absolute Elo rating levels (clusters centered around 2100s, 1200s, and 1600s). So, each cluster represents a general skill level range rather than a disparity in Elo/skill between players.

Since game outcomes appear spread horizontally across the PCA graph, the vertical slicing of clusters means that game outcomes aren't significantly influenced by general Elo rating levels alone, even if they might be affected by disparity. Instead, game outcomes appear consistently balanced across skill levels, which shows that skill level doesn't strongly sway results for specific sides (i.e. at advanced levels, white side isn't extremely advantageous compared to black side, etc…). The clustering doesn't contradict the idea that the existence of skill disparity influences game outcomes, as it doesn't assess Elo differences. Thus, our analyses hold.

The average values of the clusters are also quite interesting. When searching "lichess skill level by rating," the first result is a Lichess forum post from 2 years ago titled "What is a good lichess rating?" [1]

The consensus: "600+ is rookie, **1100+** is beginner, **1600+** is intermediate, **2000+** in advanced, 2300+ is expert and 3000+ is super-GM"

KMeans divided the players by Beginner, Intermediate and Advanced! (Though that's probably what naturally happens when you divide Elo up by 3).

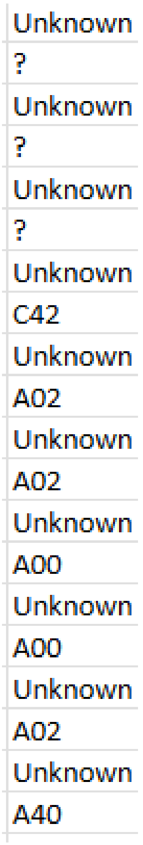
[1] https://lichess.org/forum/general-chess-discussion/what-is-a-good-lichess-rating

3. MultiD Analysis

i. Formatting

This was probably one of my more annoying ideas as parsing and editing the values was a humongous pain, but I thought "What if I turned some other columns into numeric values so I could include them in the PCA?"

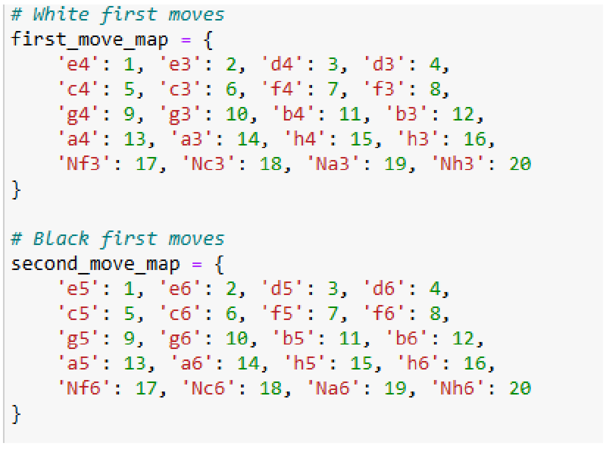
The first and easiest column to convert was eco. It looks like this:



Unknown
?
Unknown
?
Unknown
?
Unknown
C42
Unknown
A02
Unknown
A02
Unknown
A00
Unknown
A00
Unknown
A02
Unknown
A40

The question marks, I believe, represent when the opening has not been classified as an official opening and thus does not have its own ECO code (ECO Codes is a classification system for opening moves in Chess). I replaced all question marks with 0 (denoting that the players did not start off with an established opening) and the codes with 1.

As mentioned before, every other row is filled with "Unknown" besides the last column which has every single chess move played in that match. I decided to extract 2 other columns as the first white move and first black move of the match. This required a good amount of trial and error with regular expressions and brute forcing indexing with strings, but I was eventually able to extract the first move of black and white, which I then manually mapped to integers that I assigned to each of their 20 respective moves:



```
# White first moves
first_move_map = {
    'e4': 1, 'e3': 2, 'd4': 3, 'd3': 4,
    'c4': 5, 'c3': 6, 'f4': 7, 'f3': 8,
    'g4': 9, 'g3': 10, 'b4': 11, 'b3': 12,
    'a4': 13, 'a3': 14, 'h4': 15, 'h3': 16,
    'Nf3': 17, 'Nc3': 18, 'Na3': 19, 'Nh3': 20
}

# Black first moves
second_move_map = {
    'e5': 1, 'e6': 2, 'd5': 3, 'd6': 4,
    'c5': 5, 'c6': 6, 'f5': 7, 'f6': 8,
    'g5': 9, 'g6': 10, 'b5': 11, 'b6': 12,
    'a5': 13, 'a6': 14, 'h5': 15, 'h6': 16,
    'Nf6': 17, 'Nc6': 18, 'Na6': 19, 'Nh6': 20
}
```

I think these are right but also there were a few NaNs in these columns so I might've mirrored something wrong, but there weren't too many so I just dropped them so I could see the result.

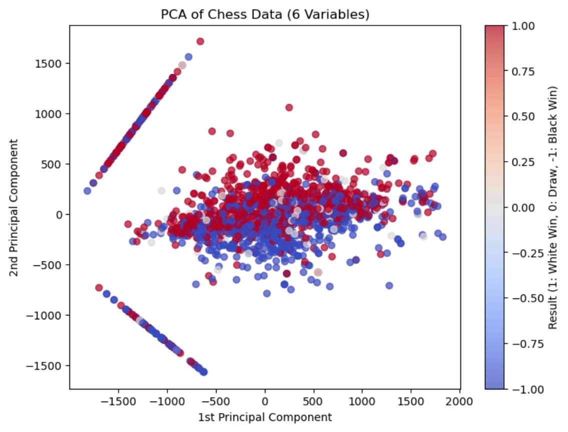
Here's what the resulting PCA-ready data looks like:

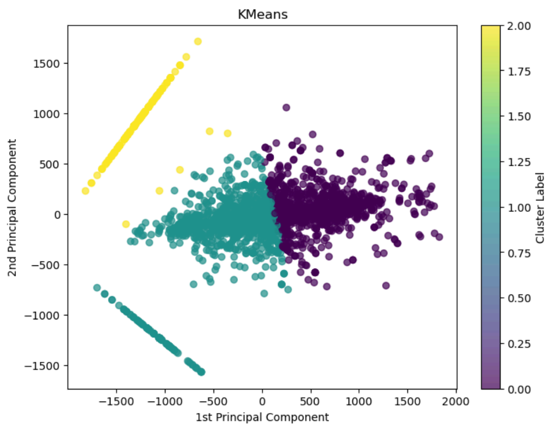
```
   black_elo  white_elo  result_encoded  eco  first_move_id  second_move_id
0     1582.0     1477.0            -1.0  0.0           17.0             8.0
2     1594.0     1577.0             1.0  0.0           17.0             8.0
4     1794.0     1708.0             1.0  0.0           17.0             8.0
6     1609.0     1457.0            -1.0  0.0           10.0            10.0
8     1470.0     1564.0             1.0  0.0           12.0            10.0
```

So, what do the resulting graphs look like?

ii. Hmmmm….

The resulting PCA and Clustering are as so:





I'm not too sure what to make of this, but I do feel like there shouldn't be those straight lines. I would assume that I made a mistake somewhere in choosing how to express the data numerically or formatting something…?

Either way I don't think I really want to pursue this side further going forward…for now. I'm more so interested in learning more about scraping, because according to Chess users on Kaggle, some of the Lichess game data also have chess engine analysis information stored in them, and thus I could get advantage values and mistake counts for each player. Some of the 6000 data points I have thus far collected could be missing a lot of rich information calculated by chess bots. I just have to figure out how to query that from the Lichess API.

Lichess itself does release data dumps every month with a bunch of games that include chess engine analysis data, but that doesn't sound as fun. However, if looking through the API does hit some unscalable walls, that will be what I look to next for mistakes and performance analysis.

4. Fun Stuff

The following sections include things that I discovered and looked into on my preliminary research on the topic of chess. They were things that I was interested in before this class that were related to the algorithmic and coding side of chess websites. Turns out, they aren't directly related to my project, but were still interesting nonetheless.

…I also took notes on them so I at least want to submit them in case someone was interested in what I found.

i. What is Elo?[2]

From what I've read online, chess has/has had 3 main rating systems: Elo, Glicko 1, and Glicko 2. Elo is the oldest out of the 3, and is relatively simple mathematically. Glicko 1 has better prediction accuracy than Elo in exchange for being more mathematically complicated, and Glicko 2 is an updated version that is even more accurate as it adds volatility into its calculations. Lichess actually uses Glicko 2, not Elo. However, since "Elo" as a word has been used so widely to describe skill rating, many people today still use it to describe skill points and rating, and thus for the future I will continue to use Elo (it's much easier to type/say compared to Glicko 2 too).

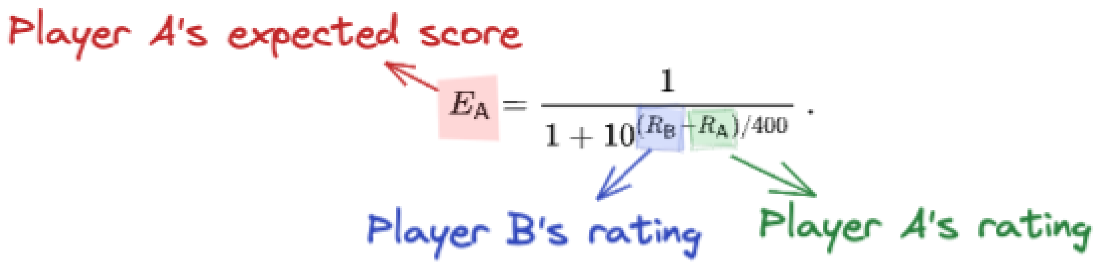
Rating systems have prediction and update components that connect with each other. Since prediction tells you the expected result of a match and updating tells you how to adjust the players' Elos after the end of the match, after a period of time, ratings should be good indicators of who is going to win/lose.

On Lichess, when you first start playing, your rating starts at 1500 +/- 1000, 1500 being your Elo and 1000 being the "confidence interval" the system has of how accurate that rating is for you as a player.

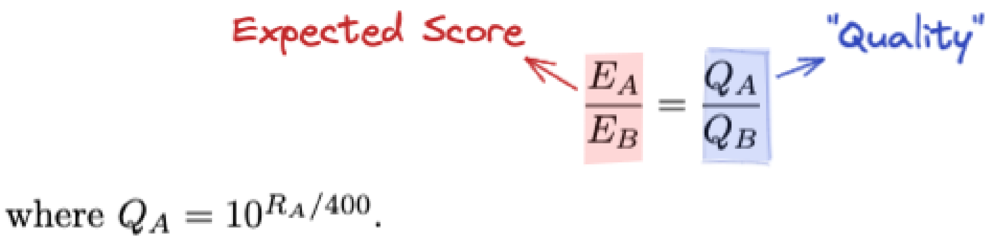
[2] https://lichess.org/faq#ratings

ii. How Elo Gain/Loss is Calculated[3]

As a result, when you first start playing, your rating will change by a lot after each game, but as you play more games, Lichess' system gradually starts to understand where you belong, decreasing the amount of points gained and lost per game. However, a problem is that though a higher rated player will usually win, they aren't guaranteed to. The Elo rating system deals with this by, in FIDE Master CheckRaiseMate's words, seeing it as "the Pokemon Trading Card Game." Both players in the chess match draw from a deck, and whoever draws the stronger card wins, but the higher rated player has stronger cards in his deck. The assumption is that each player has a normal distribution as their deck, with their current rating being the center (ex. If you are rated 1600, most of your cards are at 1600, but you have some around 1400 and 1800 as well), and thus an 1800 player could draw a 1600 and lose against a 1600 rated player once in a while. The points each player receives/loses after a match is then calculated through the players' expected scores following these formulas:



$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}}.$$



$$\frac{E_A}{E_B} = \frac{Q_A}{Q_B}$$

where $Q_A = 10^{R_A/400}$.

From CheckRaiseMate's "How Elo Ratings Actually Work"

And, after a chess game, the players' resulting updated ratings are "just the difference between the expected result and actual result, scaled by a constant" called the k-value. It's still a bit confusing to understand, but CheckRaiseMate

[3] https://lichess.org/@/CheckRaiseMate/blog/how-elo-ratings-actually-work/J8UZThlO

includes an actual example event so you can see the numbers being calculated and changing (if you somehow are still reading this progress check's endless blocks of text, I've footnoted a link to the article).

iii. Matchmaking Nuance

This section is moreso about speculation and describing community doubts surrounding how Lichess explains their rating system.

There doesn't seem to be an officially published description of how Lichess' matchmaking works; one can only assume that it attempts to match players with similar Elos and confidence intervals. Yet, despite the aforementioned explanation of the chess rating system, people are still a bit skeptical. There've been many people online throughout the years complaining about gaining too little or being matched with players too high above their rating, some even being paired up against players 500+ rating above them. If you're playing in a variant of chess that has a very small pool of players queued up, the matchmaking system will eventually pit you against someone by force no matter the rating difference. Or, if your account's confidence interval is still pretty wide, you could lose a ton of rating pretty quickly. But, how about the long-time players that are queuing for the most popular variants?

My guess is that there is a separate system/augment to the rating system that is meant to discourage "smurfs," the term used to describe users who make fresh accounts to play against lower level players for easy wins. What many competitive video games do these days is detect certain smurf-like behavior within newly created accounts and put them into a matchmaking queue called "smurf" queue. In this queue, smurf accounts (and because the system is almost always flawed, some poor actual new accounts as well) will lose a lot of rating when they lose, and gain a very small amount of rating when they win. However, they will consistently be matched against players their actual rank or higher, making it difficult for them to win.

Lichess having a system like this would explain why certain users sometimes complain about the unpredictability of Lichess' rating and matchmaking system despite the theory surrounding ratings being well documented, and also shows why what we know about its system can't entirely be relied upon. The existence of smurfs could also sway the data that we gathered—was it just by chance that a 800 rated player beat a 2500 rated player, or did Lichess' system know that would happen?

iv. What is a "Stockfish"???[4]

I kind of lied in the abstract, since this section will actually be pretty important to understanding what the heck I'm looking at later in my project. Stockfish is considered to be the world's strongest chess engine, utilizing a large database of human moves and algorithms to make and analyze chess game decisions. According to Lichess' database, about 6% of the games in their monthly released data contain Stockfish analysis evaluations. A set of moves (one from White, one from Black) in these games is formatted within a file like so:

```
1. e4 { [%eval 2.35] [%clk 0:00:30] } 1... c5 { [%eval 0.19]
[%clk 0:00:30] }
```

The data is always from White's point of view. I'm still not entirely sure what to make of this, but do my understanding, 2.35 is the percent advantage that White has after their move of e4, and 0.19 is the advantage they end up having after Black moves (White loses about 2% advantage, which isn't too surprising since there shouldn't be much advantage from either side after the first 2 moves). The percentages represent 1% of the "centipawn" advantage, meaning that White had a 19 centipawn advantage after the first set of moves were made.

So what exactly is a centipawn then? When you first begin playing chess, you are taught that each piece has its own worth relative to that piece's utility. A pawn is worth 1 point, a knight and bishop 3 points, rook 5 points, and the queen 9 points. Thus, from a very basic perspective (ignoring game state), if one player only captured a queen, and the other player only captured a rook, then the first player would be up by 4 points and thus have an advantage due to the strength of the remaining pieces on the board. The concept of a centipawn follows this concept.

"Centipawn" is a unit of measurement, one centipawn equating to 1/100 of a pawn. However, having a 100 centipawn advantage doesn't mean having 1 point up on the enemy in terms of pieces captured; centipawn measures the value a player's "board-state" has. Thus, by White making their first move in the example above,

[4] https://database.lichess.org/#standard_games

they gained enough winning-power from the positioning of their chess pieces to equate to having about 2 pawns over their opponent. How many centipawns a player gains or loses from making a move on Lichess is determined by Stockfish, which knows pretty well at this point how to calculate the adequacy of players' moves.