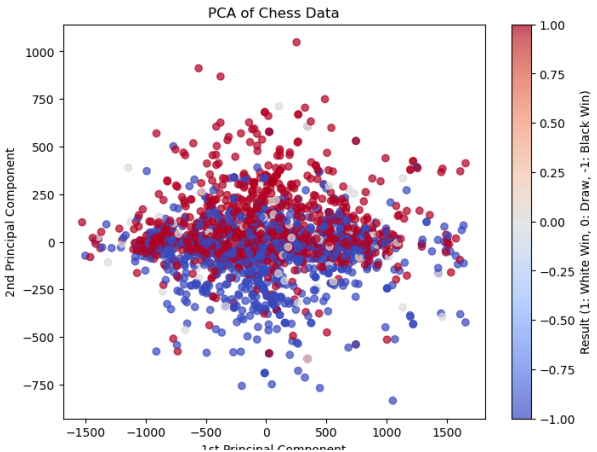
I wanted to emulate the data within Lichess datasets available on Kaggle, which means scraping the game data of a bunch of Lichess users. Much of what I've done thus far has been related to scraping Lichess user data while navigating networking limitations. To access the data, I first researched how 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 eventually figured out that you could generate your own API key to manually request data from Lichess instead of using a middle-man client.

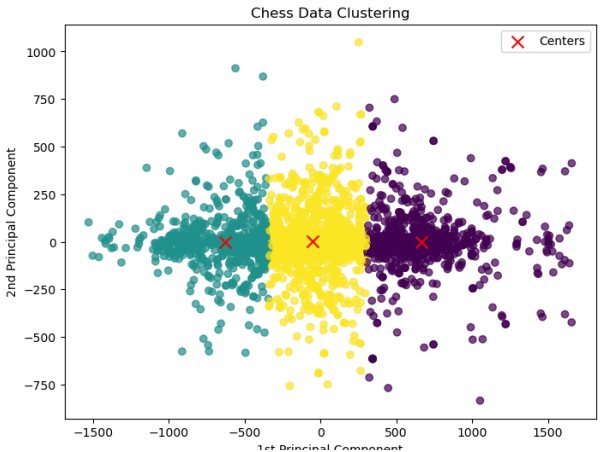
There isn't a way to just get a list of a bunch of random players, but I needed a bunch of usernames to request sample game data. However, Lichess has its own URL that you can query from with the input of a team name that returns a list of all members in a team. So, my code's general algorithm eventually became as so: Loop through a list of teams, 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). Create pauses and retries between each user to account for runtime and API rate limit errors. The resulting data looks like 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"	

For some reason, the rows come in sets of 2. The termination type is logged 2 times and covers up the "moves" column for the first row within each set, and the second row is all "Unknown" with the actual moves being logged in "moves." I decided to just manually fix it through Python and fix the scraping code later.

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.





For PCA, 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, 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. Checking the specific max and min 2nd PC values support this.

Since we only have ELO ratings and game results as our variables, these patterns would then suggest that Elo disparity influences game outcomes. On the other hand, the KMeans clusters are sliced vertically and seem to be symmetric across the x axis. Upon checking average elo ratings and game outcomes within the clusters, it can be seen that it grouped players up by elo ratings (coincidentally the same grouping Lichess uses for Beginner, Intermediate, and Advanced players). Game outcomes appear consistently balanced across skill levels, which shows that skill level doesn't strongly sway results for specific sides.

```
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
```

```
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
```

I also attempted to add more variables by mapping ECO (codes for officially recognized openings) and the first move of each player to integer values, but the resulting PCA looked very wrong so I didn't mess any further with that.

Moving forward, I'm more interested in learning more about scraping richer data, because according to Chess users on Kaggle, some of the Lichess game data also have chess engine analysis information stored in them (about 6% of all games per month), and thus I could get advantage values and mistake counts for each player through the advantage units that chess engines use to analyze each player's move. Through this, I might be able to find trends between average elo and move accuracy, or magnitude of mistakes depending on skill rating.

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.