Project

Jake Moss - s46409665

April 21, 2021

Contents

1 Introduction **1**
1.1 Project aims . . . 1
1.2 Obtaining data . . . 2
1.2.1 Sources . . . 2
1.2.2 PGN Format . . . 2
1.3 Motivation . . . 3

2 Data Processing **3**
2.1 Tools used . . . 3
2.2 Meta Data Extraction . . . 4
2.3 Extracting Implicit Data . . . 4
2.4 Data storage . . . 4

3 Data Analysis **4**
3.1 Local and Global Normalisation . . . 4
3.2 Difficulties and issues . . . 5
3.2.1 Pawn capture on 8th rank . . . 5
3.2.2 Timezones and my ignorance of them . . . 6
3.2.3 Storing game object in Data frame . . . 6
3.2.4 Kernel density plots report negative time . . . 7

4 Conclusion **7**
4.1 Results . . . 7

5 Reflection **7**

6 References **7**

1 Introduction

1.1 Project aims

The goal of this project was to provide a new and somewhat unique visualisations of chess games. I plan to do this by plotting various positions of events such as

☒ Captures

- ☐ Checks
- ☐ Checkmates

In addition to these I plan to show when these events happen throughout games.
The purpose of this project is to show the positional differences between different levels of play and over time. This is something not often visualised within the community thus I thought it would be a unique take on some common statistic.

1.2 Obtaining data

1.2.1 Sources

Chess games are found freely on the internet from many archives such as PGN Mentor. From here I am able to download thousands of games at mass using a FireFox extension. However as most preserved games are of tournaments and high skill players this data doesn't not sure what happens in common games. To help mitigate this I can use the public API's from Lichess and Chess.com to gather games from friends and players from various ELO's. To pull games from Chess.com I use this bash command to gather the PGN files from individual months and write them to a single file as Chess.com does not support downloading of all games at once.

```
for g in $(curl -Ls https://api.chess.com/pub/player/$PLAYERNAME/games/archives | jq -rc
↪  ".archives[]") ; do curl -Ls "$g" | jq -rc ".games[].pgn" ; done >> games.pgn
```

Lichess easily allows for downloading of an entire players archive at once with a simple `curl`.

```
curl https://lichess.org/games/export/$PLAYERNAME > games.pgn
```

1.2.2 PGN Format

The most common format used to store chess games is Portable Game Notation. It is a human readable plain text notation which stores individual moves rather than the board states. While this is considerably more efficient for both storage and computation it makes extracting data that isn't explicitly given difficult and costly.
The standard defines a required seven tag roster and allows for optional tags such as `WhiteElo` and `BlackElo`. Each move is stored in Standard Algebraic Notation, which describes the change in board state such as captures, checks, and promotions.
SAN comes in a varsity of formats short, long, or minimal. Each variant encodes moves with different levels of detail. Unfortunately no single variant is used in all PGN databases, although it is rare for the format to change from within a database.
A standard might look something like this

```
[Event "WCh 2018"]
[Site "London ENG"]
[Date "2018.11.09"]
[Round "1"]
[White "Caruana, Fabiano"]
[Black "Carlsen, Magnus"]
[Result "1/2-1/2"]
```

```
[WhiteTitle "GM"]
[BlackTitle "GM"]
[WhiteElo "2832"]
[BlackElo "2835"]
[ECO "B31"]
[Opening "Sicilian"]
[Variation "Nimzovich-Rossolimo attack (with ...g6, without ...d6)"]
[WhiteFideId "2020009"]
[BlackFideId "1503014"]
[EventDate "2018.11.09"]

1. e4 c5 2. Nf3 Nc6 3. Bb5 g6 ... 115. Ra8 1/2-1/2
```

Tags preface the game in `[]`, often far more than required are provided. These tags are referred to as the headers. The moves themselves are listed in SAN with the result at the end.
Due to the variation within each format and the format implicitly encoding data I wished to extract I chose not to write my own PGN parser and invalidator and instead to use an existing popular library, python-chess.
I would have preferred to use an alternative notation, `Reversible algebraic`, as it explicitly states the captured piece and its position within plain text. This would have allowed me to directly gather the data using regex without playing out each game.

1.3 Motivation

As someone who does not play chess this is a strange choice of a project.

2 Data Processing

2.1 Tools used

In this project I make use of 4 libraries

- python-chess Used to handle the complex move validation and PGN parsing required to step though games.
- matplotlib Standard plotting library.
- seaborn A high level plotting library built on `matplotlib`.
- numpy Standard scientific computing library. Within this project it is used to for its `pandas` optimisations, and reshape function.
- pandas Data frames provide an efficient way to store and conditionally select games based on metadata.

2.2 Meta Data Extraction

To extract the metadata from a game the PGN file is read using standard python methods and the game object is initialised using the `aggergate` function from the `aggergation.py` module. From the game object the headers are extracted into a dictionary and a data frame is created consisting of the headers are column titles and rows as individual games.

	SAN string	Black	White	Result	BlackElo	WhiteElo	Date	...
1	`SAN string`	`name`	`name`	`string`	`int`	`int`	`DateTime`	...
...								

2.3 Extracting Implicit Data

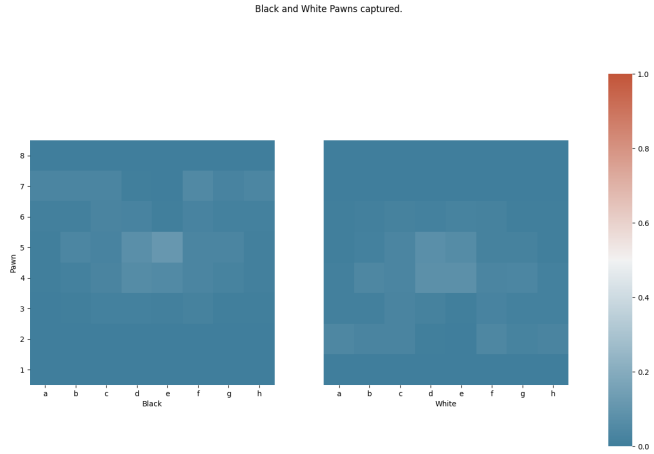
To extract the "implicit data" such as piece captures each game must be played out in its entirety with each move analysed.

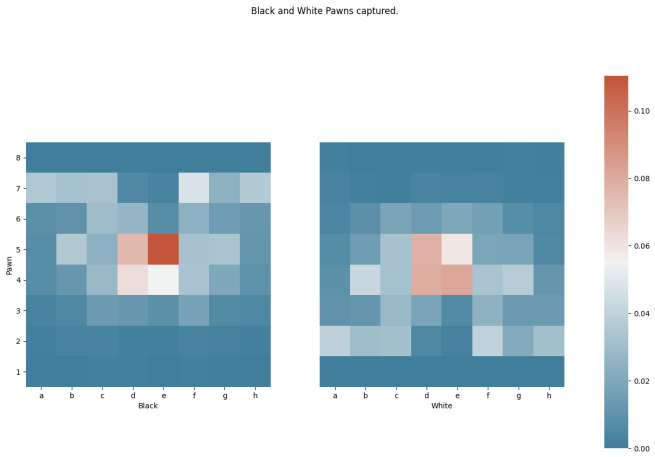
2.4 Data storage

3 Data Analysis

3.1 Local and Global Normalisation

Within the heat map grid plots, each map shares the colour bar on the right. To accomplish this each square with each map is normalised to the sum of the local grid so that the map represents proportion instead of frequency. From this we now know that the maximum value is `1` and the colour bar can be calibrated to `[0,1]`. However, as the proportion of lost pieces rarely gets close to `0.5` let alone `1`, the colours become hard to differentiate.
To remedy this I set the maximum of all local maximums to the max of the colour bar. This made it so that each plot is proportional to itself and the colouring is consistent between plots.





A similar strategy was employed to ensure the histograms and kernel density plots shared the same axis.

3.2 Difficulties and issues

3.2.1 Pawn capture on 8th rank

Some pawn capture heat maps show a capture on the 8th rank (furthest row away from the starting position). According to the FIDE

> 3.7e) When a pawn reaches the rank furthest from its starting position it must be exchanged as part of the same move on the same square for a new queen, rook, bishop or knight of the same colour. The player's choice is not restricted to pieces that have been captured previously. This exchange of a pawn for another piece is called 'promotion' and the effect of the new piece is immediate. [1]

Thus a pawn can never be captured on the 8th rank.
This bug likely occurs due to how the detection of captures and handling of positions works. While promotions are accounted for there is a specific edge case that persists.

```
def piece_delta(board: chess.Board, count: int, piece_count: Dict[int, int],
                colour: bool) -> Tuple[int, int, int]:
    piece_position = (0, 0, 0)
    for key, value in piece_count.items():
        current_count = bin(board.pieces_mask(key, colour)).count('1')
        if current_count < value: # Detects lost based on previous state
                        piece_position = (key, uci_to_1d_array_index(board.peek().uci()),
        ↪  count)
            piece_count[key] = current_count # Modify by object-reference
                            break
        elif current_count > value: # Accounts for promotion
                        piece_count[key] = current_count # Modify by object-reference
                        piece_count[chess.PAWN] = bin(board.pieces_mask(chess.PAWN,
            colour)).count('1')  # Account for pawn count change
                        break
    return piece_position # piece id, position, count
```

This code block works by making mask of the current board state. This returns a boolean bitboard consisting only of the piece requested. This is then counted and used as the number

Table 1: Piece mask of Black starting pawns

0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

of current pieces of the piece type present. As this method only keeps track of the number of pieces on the board at once and relies on the previous state to detect a capture it can be easily broken by a change in the piece count other than a capture. For example promotions, as promotions exchange a pawn in favour of another piece type the piece change is not negative. Although this is accounted for there is still a specific edge case which is not caught.
After a pawn promotes, if it is immediately captured the lost piece is still attributed as a pawn. I was unable to squash this bug.

3.2.2 Timezones and my ignorance of them

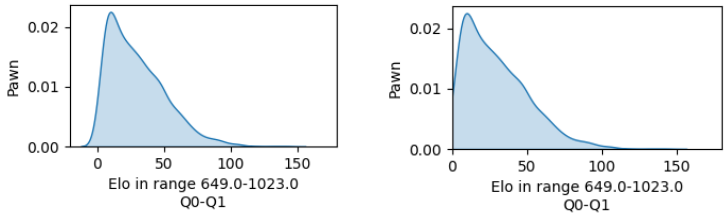
In 1918, Russia switched from the Julian calendar to the Gregorian calendar. In the switch the dates from 1st–13th of February. In doing so breaking any naive date comparison implementation from before the switch to after.
As Tom Scott put it "What you learn after dealing with time zones, is that what you do is put away from code and you don't try and write anything to deal with this. You look at the people who have been there before you. You look at the first people, the people who have dealt with this before, the people who have built the spagetti code, and you thank them very much for making it open source.". Rather than dealing with time zones the `pd.to_datetime()` method was employed to correctly handle dates. Although my use-case is not very complicated it is better to use code that I can trust.

3.2.3 Storing game object in Data frame

I ran into a lot of issues storing a game object within the same data frame as the metadata. The object would be cast to its string representation due to pandas seeing the rest of the data frame as strings. This string representation would just be the original PGN game making it very inefficient to have to prase the game again.
To over come this I stored the game objects with a normal list and made sure to sync the index of the data frame to the corresponding game within the list. As data frame indexing starts at `0` there is not offset. For example, the row of index `7` of the data frame corresponds to game of index `8` within the game list.

3.2.4 Kernel density plots report negative time

KDE plots apply smoothing to give a continuous and appealing look. Unfortunately this smooths bleeds over to the negative x-axis. This is wrong, move numbers cannot be negative, however, truncating the axis can produce jarring plots.



As the negative x-axis being incorrect and misleading I chose to truncate all histogram and KDE plots.

4 Conclusion

4.1 Results

5 Reflection

6 References

References

[1] FIDE. Laws of ches, 2008.