Report

Assignment 0: Review Unit
Chess Analyst
CSC 501

University of Victoria

Submitted to:

Prof. Sean Chester

Submitted by:

Anushka Halder

Sanjana Arora

Tavanpreet S. Oberoi

1. Introduction

Chess is a two-player, turn-based strategy game. Players make their moves by moving checkered pieces one at a time across a checkered board. There are 32 pieces in all, half of which are white, and the other half are black. One player controls the white pieces, while the other controls the black pieces. Each turn, a player may move a chess piece from one square to another. Each piece has its own set of mobility and limits, and a game concludes when one or more of the players resigns, agrees to a draw, or has no lawful move to make (i.e., no piece to move on a valid square).

The dataset used for the problem is lichess. The standard chess data for 2013-January having 121,332 game records is used for the analysis of the best suited basic data models (Array, LinkedList, HashMap) and further Exploratory Data Analysis has been performed. As observed from the given dataset that the data has been increasing rapidly with every month, it is important to design a model that supports the continuously increasing data keeping the scalability and time complexity of the model in mind.

In the below report, there is a discussion on differences between the different data models implemented, keeping the scalability factor in the process.

2. Data Modelling

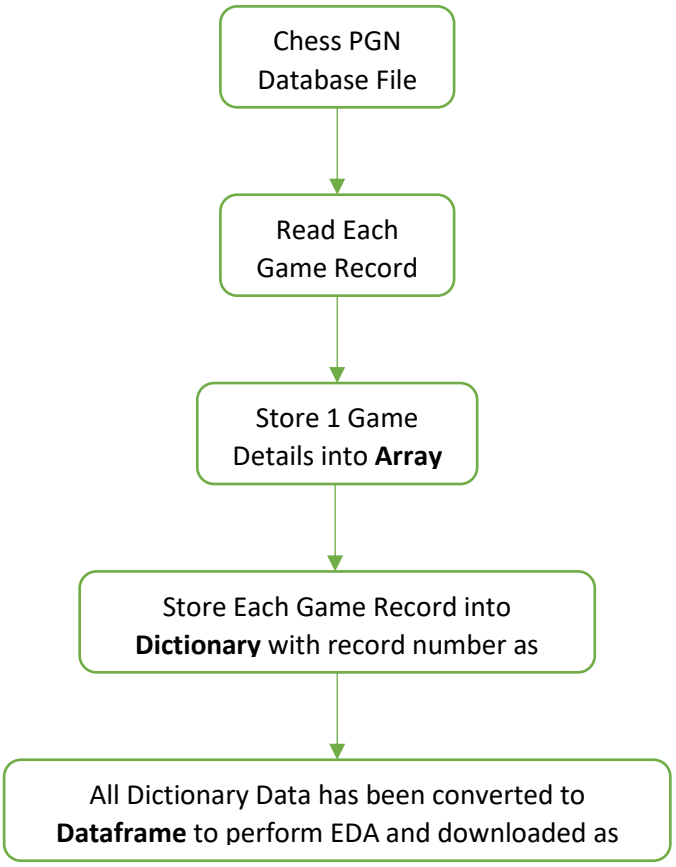


Fig.1. Flow graph for storing the data

- **Data Structure: Array**

  Data type *Array* is used to store details of single game, before pushing it to the dictionary. Some of the benefits of using Array are:

  - One array of size 16 is used to store data. This array is modified with each game row iteration.
  - As there is no processing required on the prepared data, so array can be considered good fit for this task.
  - Array stores data in contiguous format, therefore, for a single game there would not be any memory issues and improve the processing speed. However, it is not used to store all the games data, because with increasing size there could be possibility of memory timeout.
  - Selected details about the game are stored with predefined format like given in below Fig 2.

```
header_list = ['Event', 'White', 'Black', 'Result',
               'UTCDate', 'UTCTime', 'WhiteElo', 'BlackElo',
               'WhiteRatingDiff', 'BlackRatingDiff', 'ECO',
               'Opening', 'TimeControl', 'Termination',
               'MoveDetail','MoveCount']
```

Fig. 2. Game details are stored in similar order in array

- **Data Structure: Dictionary/HashMap**

  Data type *Dictionary* is used to store all the game records with 'record_id' as key and array storing details about the game from above part as value. Some of the benefits of using Dictionary is:

  - Data is not stored in contiguous manner, so less chance of memory timeout issue and more records could be saved.
  - Data accessing is faster using hashing algorithms.
  - There is not limit of storing data and any number of records can be stored, till memory is available in the system.

Challenges

- Initially tried to open database in excel but failed to do so. It is because of the limitation of the excel row size in memory which exceeds the number of rows in

the database. Alternatively, moved to access data directly in python notebook using 'Chess' library.

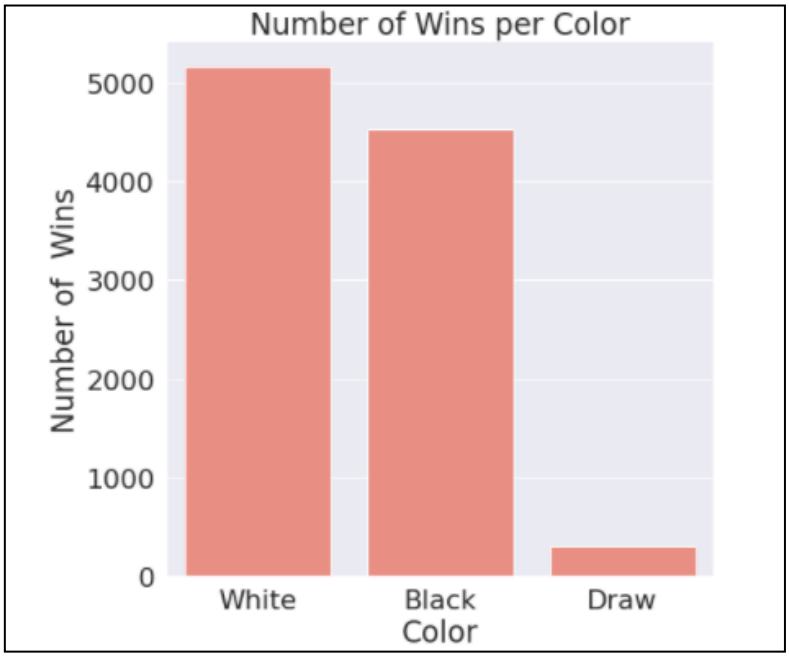
3. Exploratory Data Analysis

Visualization (Transforming Raw Data to Insights)

PLOT-1: No. of wins per color

FILES OR TABLE USED – Parsed_Chess_Data.csv
To identify the number of wins by each color, we have used the Python's pandas operation of count of unique values on 'Result' attribute. We also used pandas data processing instruction like identifying the null values to maintain the accuracy. A check for the presence of null values in the dataset was performed to maintain accuracy.

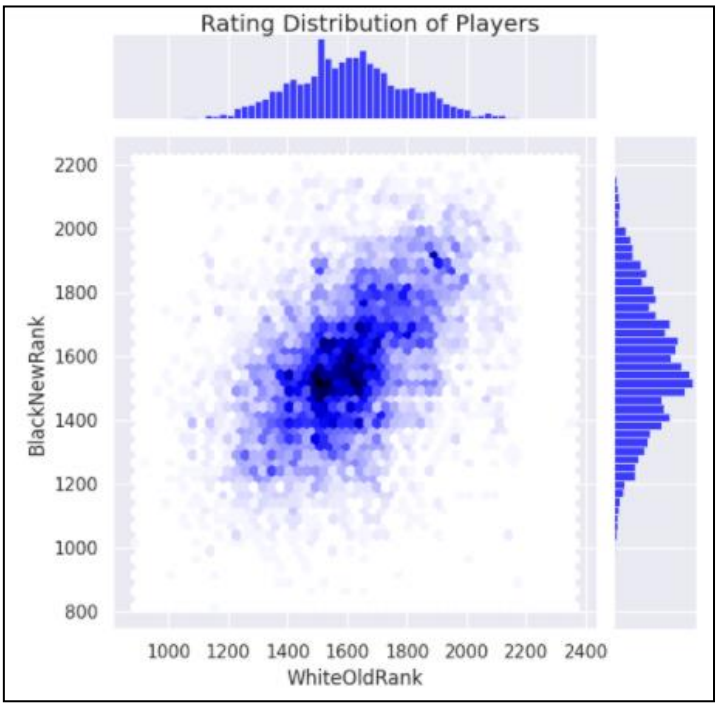
INSIGHTS – In chess community it is a known fact that the player on the white side of chess always has an added advantage in the game as the white side must make the first move of the game. Our data rightfully reflects this advantage at the white side, as the numbers of games won by the white side are higher than the number of games on the black side.



PLOT-2: Distribution of rankings of Black and White side players

FILES OR TABLE USED – Parsed_Chess_Data.csv
To understand the general ranking distribution of players playing from black and white sides, we have used the Python's Seaborn Library to plot a jointPlot of rankings of black and white players.

INSIGHTS - The jointPlot below shows the general distribution of black and white side players. As observed from the graph, the players with approximately 1500 ranking have played from both the black and white side.

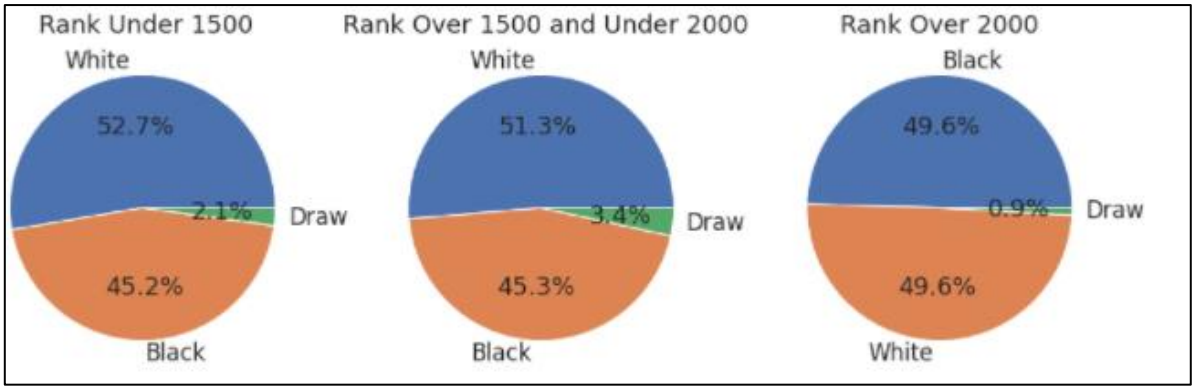


PLOT-3: Result Distribution with Ratings

FILES OR TABLE USED – Parsed_Chess_Data.csv

To understand the general rating distribution of players corresponding to the result of a game, we have calculated the mean of ratings of the players of a game and identified three player rating group from the dataset namely ranking under 1500, ranking over 1500 and under 2000 and ranking over 2000. The result distribution of color side with respected to each ranking group has been identified.

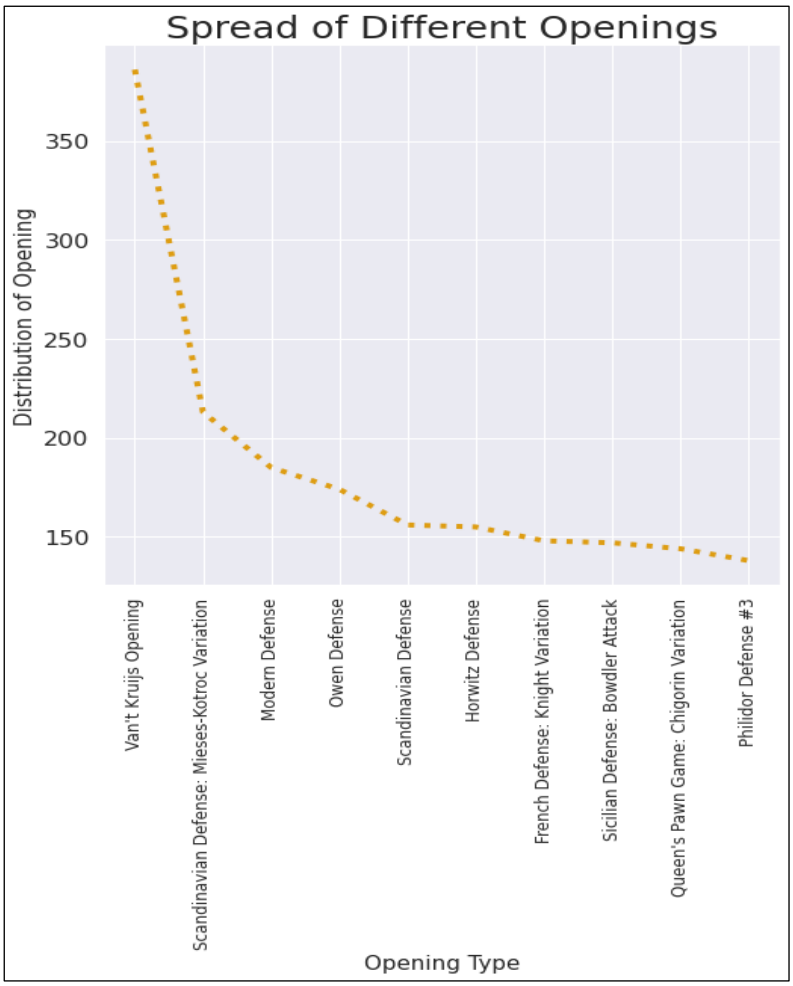
INSIGHTS - The pie chart below indicates that white side continues to win within the ranking groups of under 1500 and also between 1500 and 2000. Further, black and white sides win the game equal number of times for the players with rank over 2000. Therefore, it can be concluded that the players having ranks under 2000 play well from white side, this is attributed to the fact that these players possess more skills and they use those skills well from the white side.



PLOT-4: Distribution of types of Openings

FILES OR TABLE USED – Parsed_Chess_Data.csv
To see how the data is distributed across the various opening types in the Chess game, we have used the Python's pandas operation of count of unique values on 'Opening' attribute. In addition, we have retrieved the ten most used Opening Types for presenting on a chart.

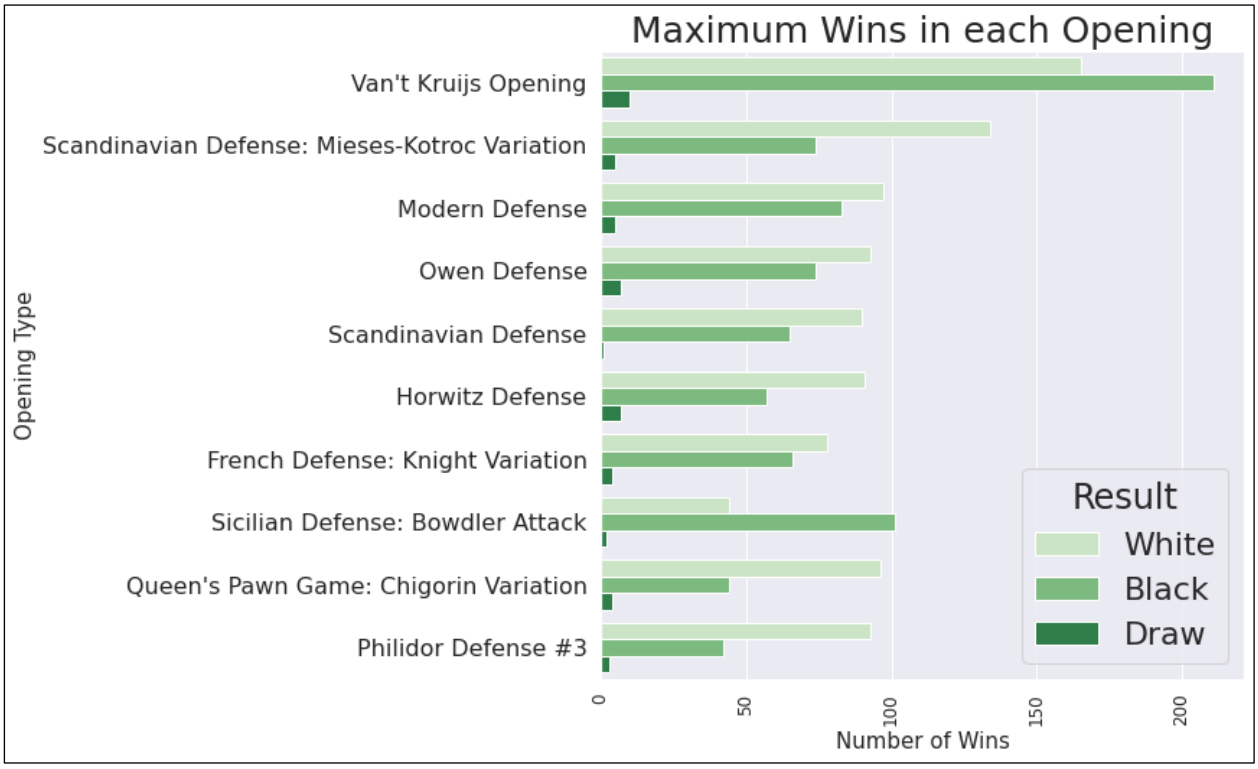
INSIGHTS – We are visualizing the top 10 openings from the obtained data to give us an idea about the opening type being used by most of the players. It can be observed that the Van't Kruijs opening is the most popular opening amongst the players. This insight will help us in comparing winners based on these openings in the next plot as discussed below.



PLOT-5: Who wins the most in each Opening Type

FILES OR TABLE USED – Parsed_Chess_Data.csv
To identify the openings used by the players to attain maximum wins/draw in different games. We are plotting a bar chart to show the comparison of the result for an opening. We are sorting them based on the 10 most popular openings.

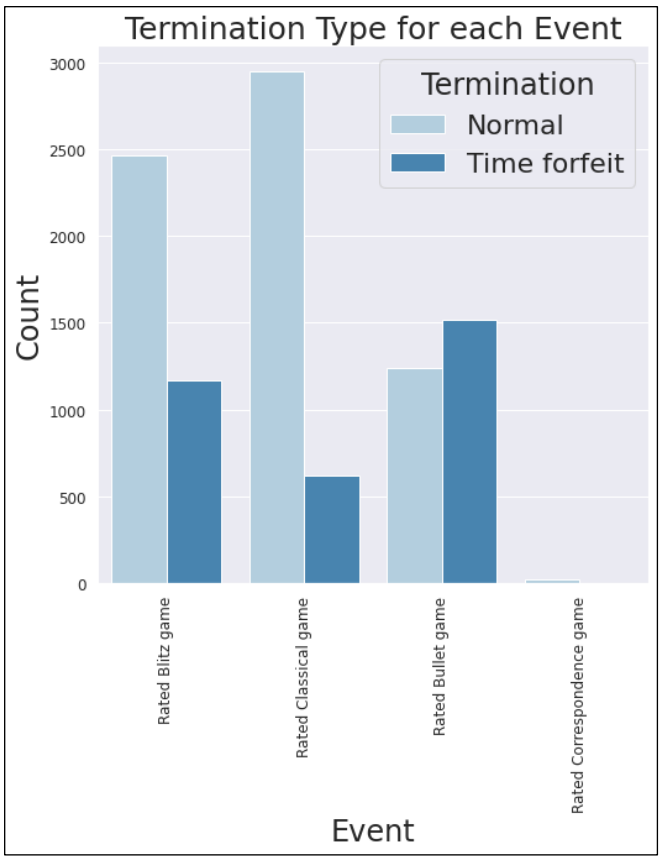
INSIGHTS – From 8 out 10 Opening types, White is winning majority of the games. However, in Van't Kruijs and Sicilian Defense:Bowdler Attack, black is taking the lead. Draw remains almost negligible compared to the wins in all the observed openings. This analysis can help players choose an opening in order to win a game.



PLOT-6: Termination Rate per Event

FILES OR TABLE USED – Parsed_Chess_Data.csv
In the below chart, we are using Seaborn library to create a plot to represent the rate of termination in each of the top 5 game categories.

INSIGHTS – We observe that in each game category, the termination is mostly normal except for Rated Bullet Game. The ratio of the termination counts is high for the top 2 game categories. This representation could in a way, also depict the difficulty level of the Red Bullet game category based on the termination.



4. Scalability

Processor – Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz 2.60 GHz

RAM – 8GB

Tool/software – Jupyter Notebook and Google Collab

Language – Python 3.7

Time taken to parse n records into dictionary:

- 0.1k games: 0.89seconds
- 1k games: 9.73seconds
- 10k games: 109seconds
- 50k games: 586 seconds

Observations:

- It has also been observed that when other task is open in the background, along with the execution, then the parsing time is more by almost 60%.

- When tried to extract data from database to excel, there was a runtime error even after larger processing time.
- There is no limit on dictionary, so more data can be stored and manipulated based on the requirement which makes the data model scalable.
- NumPy array was implemented to check the processing time, but dictionary outperformed it.
- We tried our data parsing experiments with a much larger dataset, however, there was an error in executing those experiments owing to the fact that we have a hardware resource limitation at our end. Therefore, we chose a dataset that was enough for generating meaningful and interesting insights yet scalable enough to be handled by the hashmap data structure implemented on our computer hardware. A database system such as SQLlite or a powerful hardware machine would be a very good solution to this problem.
- The hashmap allowed us to more effectively use the commonly available random memory access. Hence, making an efficient use of available memory space. Data structures such as arrays do not support the random memory access and hence, require more memory. Hashmap provided us with a scalable solution to managing large dataset.
- We have attached an .ipynb file indicating an experiment that we conducted by using array for parsing the .png file into a dataframe. The computing times for both array and dictionary are mentioned as follows:-

  Time taken to parse n records into dictionary:

- 100 games: 0.9194724559783936 seconds
- 1k games: 9.315739631652832 seconds
- 10000 games: 98.72601342201233 seconds
- 50000 games: 533.3603661060333 seconds

Time taken to parse n records into array:

- 100 games: 0.9224898815155029 seconds
- 1k games: 9.342219829559326 seconds
- 10000 games: 106.58094382286072 seconds

- 50000 games: 525.9765701293945 seconds

As can be observed from the data provided above, the parsing of .png into an array consumes more time as compared to the dictionary. Further, it was observed that parsing the data into array consumed alot more processing power and memory. That was evident from the computer performance while parsing the data into a .png file. Hence, hashmaps provided a scalable data model for the provided dataset.