

Photo by [Randy Fath](https://unsplash.com/@randyfath?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText) on [Unsplash](https://unsplash.com/s/photos/chess?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText)

Chess Winner Prediction

Scott Sigmund | 03/29/2021

# Problem Identification

Using known data can we identify the winner of a chess match 95% of the time based on player ratings, piece color, and number of turns.

## Data

The data was gathered from a Kaggle dataset that contains chess match data from Lichess.org. The dataset contains data on over 20,000 matches and 1,500 players.

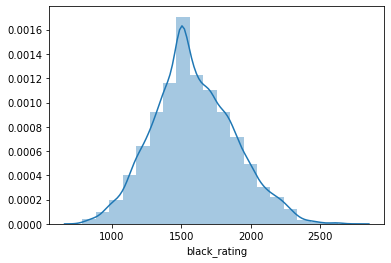
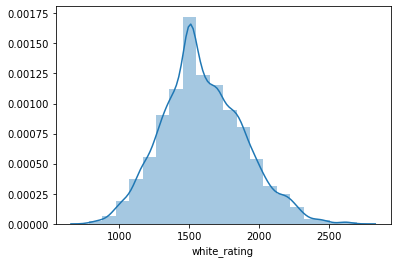
* [Kaggle Data](https://www.kaggle.com/datasnaek/chess)

# Data Wrangling

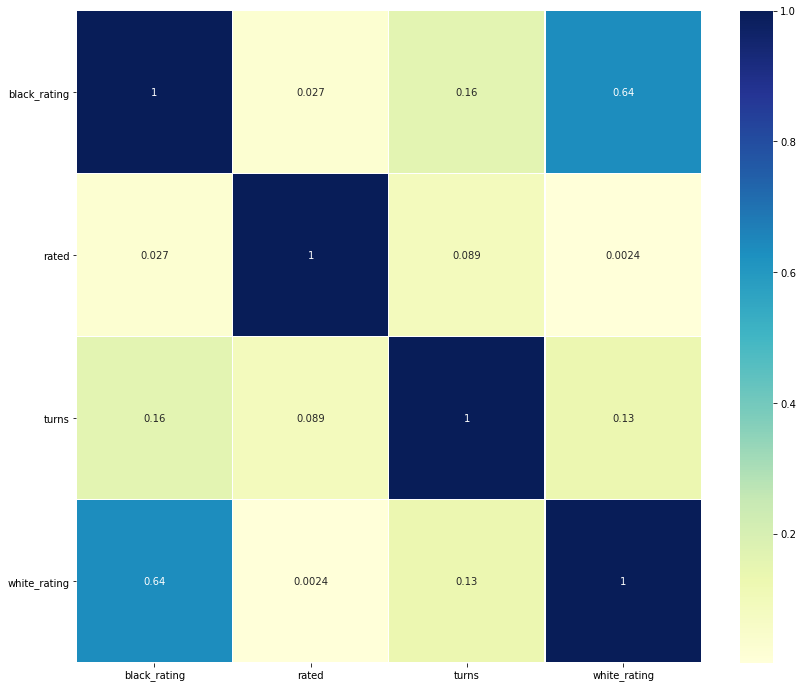
In my first review of the dataset, I pulled the data into a python notebook and view the different types of data. At first glance the data was relatively clean and did not need too much altering. My first goal was to remove columns that I planned not to use. The columns that I did not plan on using were the data that involved the specific moves made, the date of the match, the player’s Id, and increment code. The next step I took was identifying and removing duplicate records in the dataset. Once complete I exported my data to a csv file.

# Exploratory Data Analysis

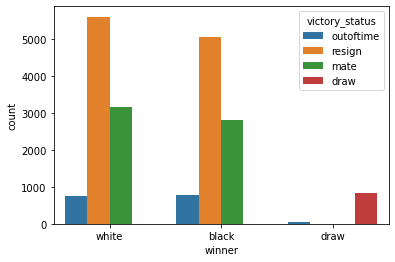
Comparison of rating of players starting as white vs rating of players starting as black



Correlation heatmap



Winner victory status bar graph



* [EDA Report](https://github.com/scsigmund/Springboard/blob/main/Project/EDA.ipynb)

# Preprocessing and Training Data Development

In the preprocessing I created dummy data for the victory status so that would allow me to set the winner column as my target field. I then scaled the data using standardscaler on the player ratings and turns and created a train test split.

* [Preprocessing Report](https://github.com/scsigmund/Springboard/blob/main/Project/Preprocessing.ipynb)

# Modeling

* [Modeling Report](https://github.com/scsigmund/Springboard/blob/main/Project/Model.ipynb)

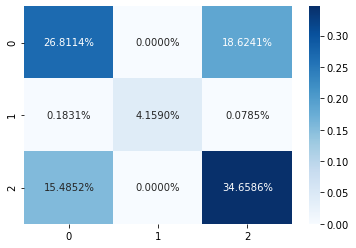
## Logistic Regression

Accuracy of Logistic Regression: 0.656290871043683

Precision of Logistic Regression: 0.6566652123791601

Recall of Logistic Regression: 0.656290871043683

F1 of Logistic Regression: 0.6557997713440461



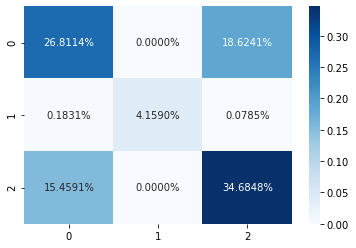
## Naïve Bayes

Accuracy of Naive Bayes: 0.656552445723254

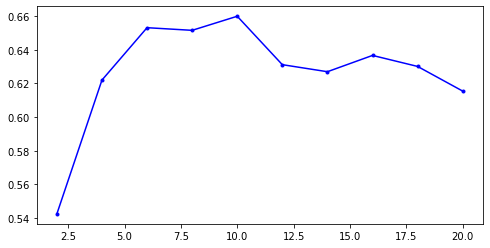
Precision of Naive Bayes: 0.6569280129710147

Recall of Naive Bayes: 0.656552445723254

F1 of Naive Bayes: 0.6560507856822165



## Decision Tree



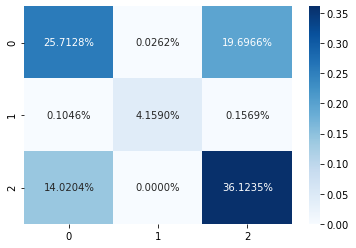
Using the above chart to select the maximum number of branches.

Accuracy of Decision Tree: 0.6599529165576772

Precision of Decision Tree: 0.6607792954885098

Recall of Decision Tree: 0.6599529165576772

F1 of Decision Tree: 0.658113711440929



# Findings

| **Model** | **Accuracy** | **Precision** | **Recall** | **F1** |
| --- | --- | --- | --- | --- |
| Decision Tree | 0.659953 | 0.660779 | 0.659953 | 0.658114 |
| Naive Bayes | 0.656552 | 0.656928 | 0.656552 | 0.656051 |
| Logistic Regression | 0.656552 | 0.656928 | 0.656552 | 0.656051 |

My findings were that the Decision tree model had the best accuracy as that would be the model that we would use.

# Later Improvements

Some improvements that we could make to the models would be:

1. Limiting the number of features that are used in the decision tree model
2. Using a random forest model
3. Using pycaret to compare more models against each other.