Chess Analysis Final Report

Andrew Fielding, Sandeep Salwan, Quangvinh Tran

December 9, 2024

Northeastern University

DS 3000: Foundations of Data Science

Dr. Eric Gerber

Abstract

This paper examines the factors that contribute to the outcomes of online competitive chess matches, including the opening used for each game, who wins a game, and how long a game lasts. Specifically, it aims to leverage regression analysis to develop an understanding of how various characteristics of a game impact the number of moves it takes for a player to win, and to use perceptron models to both a) predict which player would win a given game and b) calculate the probability of each player to win a given game. The data was collected from an API offered by Lichess–a popular online chess platform–using Python's Berserk library as an interface. The results of our analysis reveal that the features available through the API (namely player ratings, winner color, and opening played) are largely inadequate to build a regression model that meets key assumptions and has strong predictive power for game length using our simple mathematical method. The resulting perceptron models, based on player rating differentials and move count, were fairly inaccurate at predicting game outcomes. However, they highlighted that rating differential had a greater influence on the outcome compared to game length.

Introduction

Chess is a global strategy game played by a diverse range of individuals, from children to the elderly, with countless openings, tactics, and playstyles as well as rating levels for players that all influence how the match is played and ultimately, the outcome of the match. With the advent of online chess platforms, one's personal statistics and playstyle are more readily available and easily understood than ever. This allows many dedicated chess players hone in on these aspects while reviewing past games or studying the professionals in search of improvement. This project seeks to understand the relationships between numerous variables including openings, player ratings, match rating differentials, match length (number of moves), and match outcome. Specifically, we hope to leverage those relationships to answer the following key questions, which may aid players in improving their game:

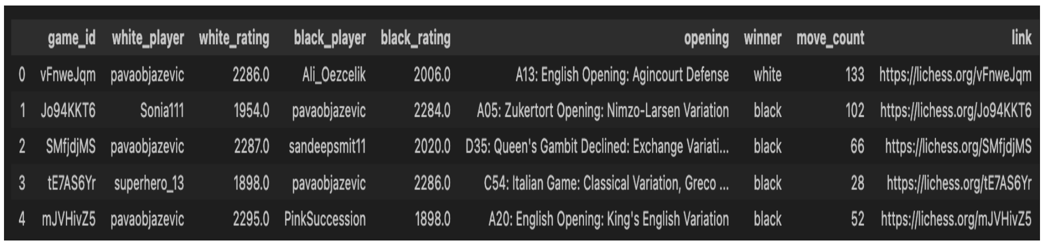
1. How does rating differential between two players impact the actual outcome of a match?
2. How can players understand and judge their probability of winning a match?
3. How do the openings the players engaged with, the ratings of the players, and the winning outcome impact the length (number of moves) of the game?

Data Description (Link to our dataset[1])

To address the proposed research questions, the Lichess API was utilized to extract data from matches from the online chess server, Lichess.org. The resulting data set contained data from nearly 160,000 matches with player ratings. The final csv data file contained information on a match's game id, white player username, white rating, black player username, black rating, opening, winner, move count, and a link to the match.

Pipeline Overview:

Lichess API provided us the data. To interact with Lichess we used a Python library named Berserk. We obtained rating differences between players, standardized ratings, and categorized ratings into groups to understand variations in performance across skill levels. This data also reflects moves postopening and primarily retains the most common openings. Thus, through compiling a large number of games, it was easier to study patterns in players' performance, results, and strategies by skill level. We used top players from Lichess leaderboards and looped through each player's games to pull individual game observations. The API was effective when it came to acquiring structured game data, especially for moves and results. This method helps make the data acquisition process quicker and also guarantees that all the data used is reliable.



	game_id	white_player	white_rating	black_player	black_rating	opening	winner	move_count	link
0	vFnweJqm	pavaobjazevic	2286.0	Ali_Oezcelik	2006.0	A13: English Opening: Agincourt Defense	white	133	https://lichess.org/vFnweJqm
1	Jo94KKT6	Sonia111	1954.0	pavaobjazevic	2284.0	A05: Zukertort Opening: Nimzo-Larsen Variation	black	102	https://lichess.org/Jo94KKT6
2	SMfjdjMS	pavaobjazevic	2287.0	sandeepsmit11	2020.0	D35: Queen's Gambit Declined: Exchange Variati...	black	66	https://lichess.org/SMfjdjMS
3	tE7AS6Yr	superhero_13	1898.0	pavaobjazevic	2286.0	C54: Italian Game: Classical Variation, Greco ...	black	28	https://lichess.org/tE7AS6Yr
4	mJVHivZ5	pavaobjazevic	2295.0	PinkSuccession	1898.0	A20: English Opening: King's English Variation	black	52	https://lichess.org/mJVHivZ5

Figure 1: Preview of the first 4 four rows of our dataset

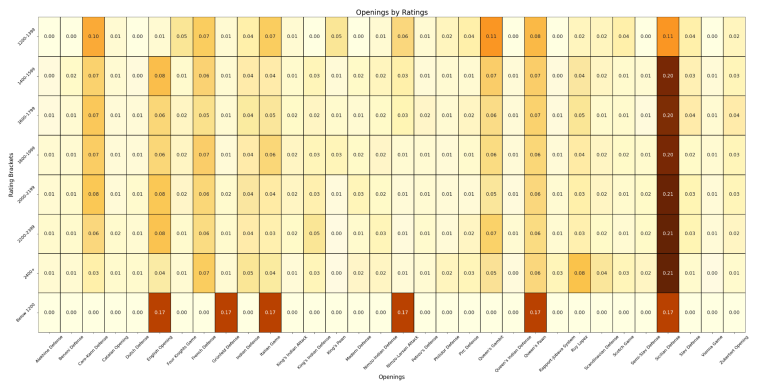
[1] https://github.com/sandeepsalwan1/ChessAnalysis/blob/main/lichess_games_data.csv

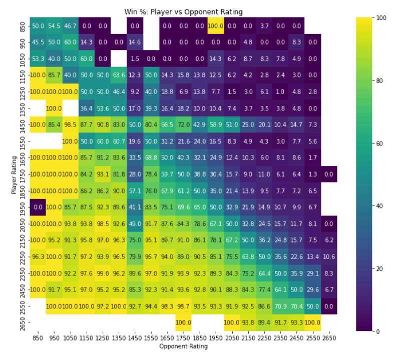
Preparing the data for our model:

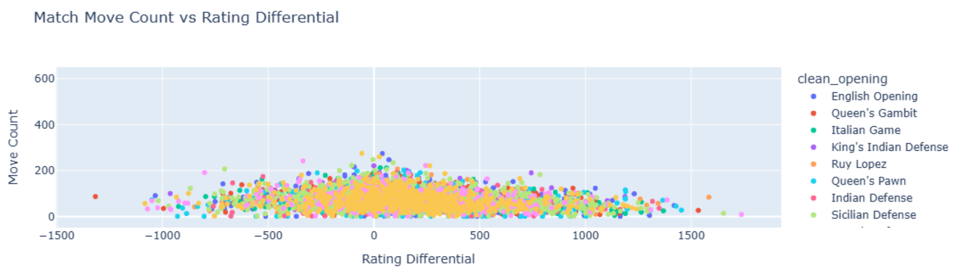
First, we retrieved the raw data and then we did basic cleaning, deleting games which have no opening, deleting incomplete records, and formatting the features properly. This resulted in a clean CSV file. Afterwards, we cleaned and integrated the data to a format that is analysis-ready. We made rating categories, calculated rating differentials, computed average ratings of players, and encoded the winners as numerical targets. Also, we used data-driven feature engineering, for instance, standardizing the ratings and encoding openings as dummy variables to make them usable for modeling. The final dataset contained ratings, openings and results that were structured in a way that made it possible to train regression or classification models to determine game aspects like whether rating differentials or opening systems affect game duration/result.

Initial Visualizations:

With over 100 unique openings in which many were not frequently used, the first visualization (fig. 2a) was a heatmap of the openings mapped onto each rating category for average match ratings. This allowed for the determination of the most frequently used openings for each rating category which could be used in future analyses. For online chess matches, players tend to be matched with those of similar skill level. For the second visualization (fig. 2b), a heat map was built to explore the win percentages for a player given their opponent's rating. It was determined that evenly rated games had closer to a 50% win probability. Facing much higher ranked players drastically decreased the chances of winning and vice versa. The third visualization (fig. 2c) was a scatter plot of the length of a match (move count) as a function of the rating differentials between the players of each match color coded by the opening used. This allowed for identification of trends for how the rating disparity between players affect how long a game might last for a given opening used. For the most part, it appeared most games ended with a similar number of moves played. As players got more evenly matched, there appeared to be a slight upward trend in the number of moves suggesting evenly rated games tended to last longer.







Clockwise from top left: Figures 2a, 2b, and 2c

<u>Method</u>

Broad outlines: We wanted to find a model that would best predict our data output. First we tried to use linear regression because we had a lot of features like player ratings, opening strategies, etc. and wanted to find the best. Multiple regression helps us see if changes in rating differentials or opening selected influentials move count (game length). Number of moves varies a lot so regression helps us see the average and allows us to view the pattern. The formula follows: $(y) = \beta_0 + \beta_1 \times (\text{Rating Difference}) + \beta_2 \times (\text{Is\_Specific\_Opening}) + \ldots + \varepsilon$, where ε is the prediction error and y is the outcome.

Linear perceptron allows us to predict binary (yes/no) values such as the outcome of a match (white win/lose). Perceptron takes in a bunch of inputs(rating differentials) and draws multidimensional boundaries separating white and black wins. The separation line's primary goal is using this model which happens when the model draws hyperplanes (multidimensional lines) and continues moving them around until two outputs are sorted.

Mathematics and assumptions:

Regression - Multiple regression uses multiple predictor values to predict output. If we have lot of predictor values like X = (X_1, X_2, …, X_p). The formula is Y = β_0 + $\beta_1 X_1$ + $\beta_2 X_2$ + … + β_ p X_p + ε. Our main goal is to choose the right coefficients for β_0 … β_p so that we reduce the the sum of squared residuals where each residual is (actual_y - predicted_y). More simply, we calculate the line or plane minimizes the sum $\sum_{i=1}^{n}(y_{\text{actual}_i} - y_{\text{predicted}_i})^2$. We assume for regression that there is a linear relationship between y (in our case, move count) and x (predictors like rating differential). We need to use OLS which is like finding the best straight line (or, in the case of multiple regression, plane) through points. We would need to assume that the errors are independent and have some kind of randomness. E.g. Imagine a chess player predicts game moves count 30+ times and then they get tired so the errors in predicting previous games happen with later games and so forth. We assume constant error variance so scatter of data around the line is consistent. We assume errors follow the bell curve. After these assumptions, we will have a powerful model to determine changes influenced by rating differences or openings. OLS allows us to make predictions well. So we had to manually verify all of these assumptions so our model will be good and unbiased.

Perceptron- For the perceptron we have separate assumptions. We need to determine binary results. The math the perceptron uses to find hyperplanes with weight vector w and bias b. It uses simple formula w*x + b where w * x is a dot product so we can combine weights and inputs. If w*x + b >= 0 white wins, otherwise white loses. For example let's say x is rating differential, w is importance of rating difference, b is learned baseline and white normally wins more than black so it is positive. x=150, w= 0.5, b = 50 Then 150*0.5 + 50 = 125. The algorithm constantly updates w and b when a point is misclassified. These updates change the hyperplane until both win/lose are separated as best as possible. For every iteration if the predicted value does not equal the actual value we change w(weight) and b(baseline). We run these 2 equations: w=w+η⋅(y−ŷ)⋅x. The second equation is b=b+η⋅(y−ŷ). Where η, the learning rate, is about 0.01 since there are over 100,000 data points, this is very small ŷ : predicted value. We continually repeat this loop until our points are correctly classified. *Perceptron assumptions*- We assume that the data can be separated by a straight line, and the outputs are binary (e.g. white wins or loses).

Pitfalls**:** Linear regression requires that data is linear and chess has many different possible factors and predictors so it is hard to say that these outcomes will create a strictly linear result. For example, higher rating differences could create extremely short games (like if a player constantly gets 2-move or scholar mated). We assume that we will constantly check the residuals and assumptions to make sure this model is being used properly. The perceptron assumes that only one straight line will properly separate both classes and since we had so many data points, overlapping data points make linear separability impossible. The perceptron will have lower accuracy and our data needed much more advanced methods and features to properly work with the model, but it was not as accurate as we would have hoped.

Why sound/appropriate: Regression helps predict move counts and perceptron classifies outcomes efficiently. Linear regression is simple and straightforward (X^T * X)^−1 * X^T y. We have a large dataset so we can quickly scour the games and find trends, this is also a closed-form solution which means we can answer questions directly . We also have interpretable coefficients so we can clearly see which predictors have a role in output. For example, we can check on of our predictors $\beta_{\text{rating difference}}$ = -0.3, then every 100-point increase in rating differential shortens game by 100*-0.3 or 30 moves. The perceptron is also great for simplicity which allows us to train and test rapidly and we can find essential patterns. We can get very interpretable results providing great outcomes. Overall, our models are simple and interpretable and we can understand chess relationships.

Results

To implement our regression model, we built two helper functions leveraging Numpy to do mathematical calculations "by hand". The first (line_of_best_fit()) calculates the coefficients of the line/plane of best fit for a given single- or multi-dimensional array of X values and an array of y values. The second (linreg_predict()) uses the coefficients to calculate predicted y values and residuals, as well as measures of the model's precision. Please see the full Jupyter Notebook to review the functions.

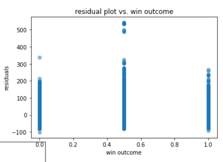
The multiple regression model implemented to predict move count ultimately included 140 X features (independent variables), including the win outcome, scaled versions of the white player rating and black player rating, an interaction between the rating variables, and dummy variables for each unique opening in the dataset. We tested various other combinations, including polynomials with some X features and other interaction terms, and found this to be the best combination.

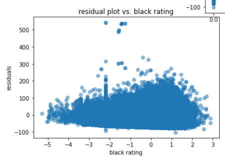
To understand the explanatory power of our model, we calculated both R^2 values and mean squared errors. R^2 provides a measurement of the proportion of the variance in the target variable (y, in this case, move count) that can be explained by the model. Mean squared error measures how close the hyperplane of best fit comes to each observation, on average. It depends somewhat on the scale of the X features, but in general a larger MSE indicates that the plane of best fit is far from many observations. Therefore, the hyperplane of best fit is the one that minimizes the MSE. We expect to see lower R^2 for the cross validated model, although the cross validated value typically better reflects the model's actual predictive power for new (unobserved) data. The results were as follows:

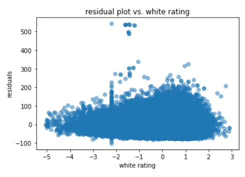
Table 1: Cross validated and full dataset results of the multiple regression model

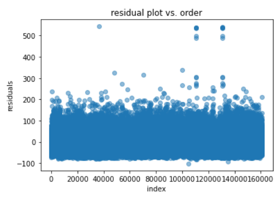
Cross validated R^2	0.064121
Full model R^2	0.067774
Cross validated MSE	1247.827118
Full model MSE	1250.291722

To understand the relative validity of the model's various assumptions pertaining to linearity, homoscedasticity, no autocorrelation, and normality, we created various plots using the residuals from the full model. This included plots of the (non-dummy) X features versus the residuals, an ordered *e* plot (residuals versus their order in the index of the dataset), and a histogram of *e*. These are standard results plots for any regression model. Any patterns in the plots of X features versus residuals suggest violations of the linearity and/or homoscedasticity assumptions, while any pattern in the ordered *e* plot indicates a violation of the assumption of no autocorrelation. Anything other than an approximately normally distributed histogram of residuals similarly suggests a violation of the assumption of normality. The plots are included below and further discussed later in the paper.









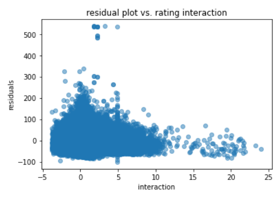


Figure 5. Residual plots for multiple regression model.

Due to the low R^2 of our original model and the patterns observed in the residual plots against the model's X features, it was suggested that we consider transforming the move count (response) variable by taking the square root or logarithm of y before implementing the model. This is a technique that is often used to correct violations of assumptions[2], and can yield a better model. The R^2 and MSE results for the transformation implementations were as follows:

Table 2: R^2 and MSE for square root (left) and logarithmic (right) transformation of move count values.

Cross validated R^2	0.106298	Cross validated R^2	0.532341
Full model R^2	0.113906	Full model R^2	0.586930
Cross validated MSE	4.548875	Cross validated MSE	25.603022
Full model MSE	4.571003	Full model MSE	25.868996

We created the same sets of residual plots to check the assumptions of the models with the transformed *y* vectors. The plots associated with the logarithmic transformation exhibited clustering of the residuals around three distinct values, and each of the x feature plots retained somewhat similar patterns to the ones seen in the plots above. The residual versus black rating plot is included to demonstrate the clustering behavior.

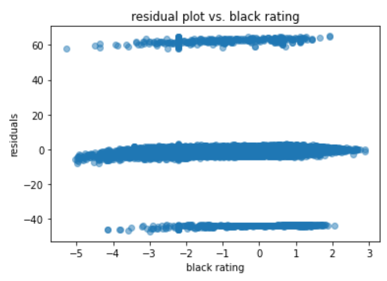


Figure 6: Example residual plot from the log transformation model

The plots associated with the square root transformation were extremely similar to those from the original regression, although the histogram was less skewed and the residual scatters appear slightly more random. The histogram and residuals versus black rating plots are included to demonstrate these assertions.

[2] See a detailed explanation of how and why this can work here.

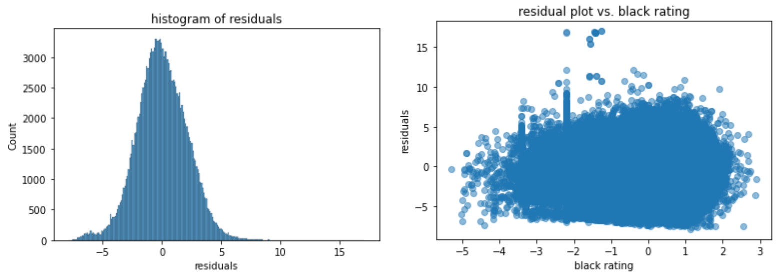


Figure 7: Example residual plots from the square root transformation model

A linear perceptron model using the hinge loss function [**eq. XX1**] was developed to predict the outcome of a match and a linear perceptron using the log loss function [**eq. XX2**] was developed to predict the probability of the white player winning the match. The independent variables for the models were the match length (*x2*) and rating differential (*x1*) between white and black (white rating - black rating) which were both standardized. The dependent variable (*y*) was the outcome of the match with white winning being 1 and black winning being 0. Single-fold 70/30 cross-validation, given the large size of the dataset, was performed to assess the performance of the perceptron models. Based on accuracy scores, it was determined that 1500 iterations were sufficient to determine the best weight vector ($\vec{w}$) for both hinge loss and log loss models. Fitting the full dataset to the optimized perceptron model to yielded the final weight vectors, accuracy scores, and lines of best fit with x2 in terms of x1 (table 3, table 4).

$$L(\vec{w}) = max(0, \; - y_i \vec{x}_i^T \vec{w}) \quad \text{eq. XX1}$$

$$L(\vec{w}) = - y_i log(\hat{y}_i) - (1 - y_i) log(1 - \hat{y}_i) \quad \text{eq.XX2}$$

Table 3: Final weight vector ($\vec{w}$ = [bias term, x1, x2]), accuracy, and equation for the line of best fit of the hinge loss function linear perceptron. x2 = match length; x1 = rating differential

Final weight vector (w)	Accuracy (%)	Line of best fit
[-0.01488, 1.3899, -0.10052]	36.54	x2 = -0.15 + 13.83x1

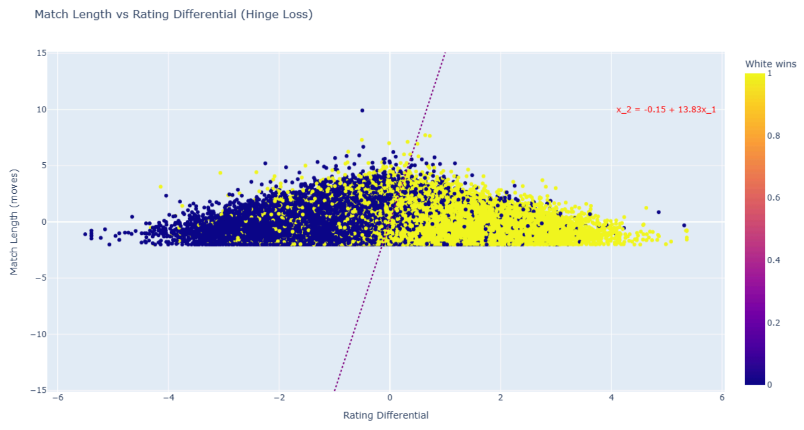


***Figure 8**. Scatter plot of match length (number of moves) as a function of rating differential grouped by white winning (yellow) and black winning (blue). A hinge loss linear perceptron was calculated and the line of best fit (purple) was superimposed over the scatter plot.*

Table [XX2]: Final weight vector ($\vec{w}$ = [bias term, x1, x2]), accuracy, and equation for the line of best fit of the log loss function linear perceptron. x2 = match length; x1 = rating differential

Final weight vector (w)	Accuracy (%)	Line of best fit
[0.00088, 1.33043, -0.08832]	27.48	x2 = 0.01 + 15.06x1

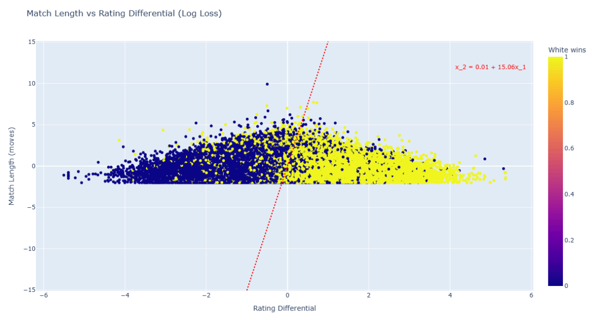


Figure 9. Scatter plot of match length (number of moves) as a function of rating differential grouped by white winning (yellow) and black winning (blue). A log loss linear perceptron was calculated and the line of best fit (red) was superimposed over the scatter plot.

Discussion

The project began with curiosity about predicting game outcomes. As a result, our first attempt to build out a model was a jury-rigged regression model to predict win probability given the differential between the ratings of the players in a game. We calculated the rating differential by subtracting the white player's rating from the black player's rating, and then engineered features by creating "bins" of rating differentials (e.g. all games with a differential of -1000 to -950) and calculating the percentage of games within each bin where the white player won. We then linked the win percentage for a bin to its median value (e.g. for the example bin, -975), and used the median values as the X feature to predict the win percentage. This yielded a prediction for the percentage of games with a given rating differential that a player should expect to win.

The model performed exceptionally well, achieving a cross validated R^2 value of 0.88. However, it ran into serious concerns pertaining to violations of assumptions, with clear patterns and a non-normal distribution discovered in its residual plots. Some of the assumptions concerns were partially alleviated by implementing a sixth degree polynomial regression using the same data–which achieved an even better cross validated R^2 of 0.99–but at that point, we ran into overfitting concerns. Additionally, upon further reflection on the usefulness of the model, we decided that it actually was not that informative. By binning the rating differentials and predicting on the average win percentage for those bins by using their midpoints, we were trying to use an average outcome for a range of data to predict a specific outcome for specific datapoint. This doesn't make sense empirically, and likely contributes to the model's astounding performance. While the model is not completely useless–it certainly helped us build a better understanding of our dataset–we decided that the win outcome would be better modeled as a binary classification problem (1 or 0 for win or loss), which could not be solved with regression. Therefore, we pivoted to the perceptron models described above and below, which are more equipped to predict categorical features like win/loss.

As we were already prepared to run a regression analysis and had built the necessary functions to create regression models, we went ahead and implemented regression to analyze how long (in terms of number of moves) a game lasts as described in the results section. After considering all three of the main implementations we tried (e.g. the original version and the two implementations with transformations of move count), it seems likely that the model with the square root transformation of *y* is the best of these three models, although all of them are significantly flawed. Directly comparing just the R^2 scores across the three models to decide which is best without considering them holistically would be a mistake, as R^2 does not fully assess goodness of fit and will be inappropriate for at least one of the models[3]. Therefore, we consider both the R^2 and the context provided by the residual plots. While the R^2 for the model predicting on log(*y*) is the highest at over 0.53 on a cross validated basis, the residual plots demonstrate that the key assumptions for regression are clearly and blatantly violated by the model. Anything other than a random scatter of points across 0 for all levels of x on the x feature plots, and for the full index on the ordered *e* plot, indicates issues with the homoscedasticity, linearity, and no autocorrelation assumptions. The plots reveal that, not only do the residuals follow patterns (e.g. systematically increase or decrease) on the x feature plots, but they also include lines of points across three different values–0, as well as approximately 60 and -50–as is clearly demonstrated in the results section. The histogram also has a large peak at 0, as well as small peaks at about 60 and -50, demonstrating issues with the normality assumption as well. Given these issues, we ruled out the logarithmic transformation model.

This left us with the original (non-transformed) model and the model predicting on the square root transformation of move count. Both models also have issues with their assumptions. Beginning with the original model, it is clear that the residuals are systematically increasing as both black and white player rating increase, before systematically decreasing once rating is near its maximum. For the rating interaction term, the opposite is true–as the x feature increases, the residuals systematically decrease. The win outcome x feature seems to be approximately random, although it does seem as though draw outcomes may have systematically higher residuals. These observations all point to this model having issues with its linearity and/or homoscedasticity assumptions as well. Meanwhile, the ordered residual plot in this case does appear to show a fairly random scatter of residuals about 0, albeit with some outliers. This suggests that the no autocorrelation assumption likely holds in this case. Finally, the histogram of residuals is *approximately* normal, but it is evident that it has a non-trivial amount of right skewness, with a longer tail to the right. The normality assumption may hold, but it is certainly not beyond doubt.

[3] For a thorough explanation of why naively comparing R^2 scores across these three models is likely inappropriate, see this page on StackExchange.

The intercept for the hyperplane of best fit calculated for the cross validated model was 75.88, suggesting that a game with all X features equal to 0 (which is highly unlikely–if not impossible–to observe empirically) would last for about 76 moves. The coefficients on scaled white rating, scaled black rating, and the interaction between them were 4.13, 5.26, and 2.66 respectively. This suggests that games including higher rated players, and especially games between higher rated players, tend to last longer, although these coefficients are difficult to directly interpret in terms of how much each additional rating point adds to a game's length since the data is scaled. The coefficient on win outcome (where 0 represents a black win, 0.5 represents a draw, and 1 represents a white win) was -3.36, suggesting that a white win is associated with a decrease in move count of 3.36. Coefficients on the multitude of dummy variables for openings varied widely, but many had a significant effect on the predicted move count. For example, the opening dummy with the largest coefficient was the Guatemala Defense (with a coefficient of about 62.6) while the opening dummy with the smallest coefficient was the Zukertort Defense (with a coefficient of about -56.05). This means that some openings (like the Guatemala Defense, which adds 62.6 moves to the expected count) were predicted to make games significantly longer, while others (like the Zukertort Defense, which subtracts 56.06 moves from the expected count) were expected to make games significantly shorter. Indeed, it is clear that–in our model–the opening often has a much larger impact on the move count than the involved players' ratings or which player wins the game. The cross validated R^2 score for the model was about 0.0641, which represented a significant improvement from other versions of the model we tested with different x features (for example, adding the rating interaction term boosted the R^2 0.571, an improvement of about 12%), but is still a low score. It suggests that the model can explain approximately 6.4% of the variation in move count using the chosen x features.

Moving on to the square root transformation model, we observed that the residual plots appeared to follow relatively similar shapes to those that we created for the original model. However, they appeared to show at least slightly more randomly distributed scatters of residuals versus x features. An example is provided in the results sections, and all graphs may be viewed in the project's final Jupyter notebook. Additionally, the histogram of residuals, while still possibility *slightly* right skewed, clearly has a shorter tail and appears more normally distributed and better centered around 0 than the histogram created in the original model. Therefore, while the model still does not perfectly meet the key assumptions, it seems to be slightly better at adhering to them than either of the other two models. Because it is predicting on the square root of move count, it is important to note that it's predicted *y* values would need to be squared to be interpreted as predicted move count (rather than square root of predicted move count), a process known as back-transforming[4]. Additionally, because the model uses a transformation, its coefficients are less interpretable, although it seems based on their signs and magnitudes that the same general trends hold. Games involving higher rated players (especially two higher rated players) tend to be longer, games won by white tend to be shorter, and the different openings have varying impacts on the predicted move count, some positive and some negative. Crucially, this model achieves a cross validated R^2 score of 0.106, suggesting that it can explain approximately 10.6% of the variability in the square root of move count using the provided x features. Therefore, the square root transformed model seems to both a) best fulfill the key assumptions for a valid regression model, and b) provide the higher R^2 score.

Although it may be the best model, these results likely still should not be accepted at face value, as there are still some issues with the model's key assumptions. Additionally, while the 0.106 R^2 score is better than the original model's score, it still suggests that the model's predictive power is fairly weak. It can only explain about 10% of the variability in the data. Thus, while we may have found the beginning of an answer to our question about what factors impact the length of a game of chess, more exploration is needed to complete the model. The next course of action to improve the model might be to examine the logarithmic transformation of *y,* since it improved the R^2 so much. There may be other ways to alter, smooth, and transform the data that fix the serious assumption issues our residual plots uncovered. Additionally, incorporating more game data–such as player move accuracy, a metric we were unable to access–might uncover new X features with more explanatory power. Overall, while the model has severe limitations, it seems to be a good starting place to fully understand the factors contributing to the move count of a game of chess.

The results from the hinge loss function perceptron indicate that the rating differential of a match has a greater impact on the outcome of a match than the length of a match. The coefficient of the rating differential, 1.3899, was 13.83 times greater in magnitude than the coefficient of the match length, -0.10052 (table 3). The accuracy of the hinge loss perceptron model was calculated to be 36.54% (table 3). The results of the log loss function perceptron model supported the findings from the hinge loss function. The coefficient of the rating differential, 1.33043, was 15.06 times greater in magnitude than the coefficient of the match length, -0.08832 (table 4). The accuracy for this model, 27.48% (table 4), was determined by setting a threshold at probabilities over 50% being a 1 and those below being a 0. Fitting the line of best fit of each model to the scatter plot of data shows the line running along a region of the graph where there appears to be mostly black wins on one side and mostly white wins on another (fig.8, fig. 9) as expected.

Overall, both perceptron models had accuracy scores that were less than 40%, suggesting they were fairly inaccurate. This was expected; however, given the numerous variables and complex nature of chess. It would be unreasonable to expect two features of a chess match to accurately predict the outcome of a match. Both models produced similar results as their weight vectors indicated having a higher rating increased the probability of winning a match and longer matches increased the probability for black to win. Additionally, the models indicated that the rating differential had a relatively larger influence on the game outcome compared to the match length. Future analyses using a perceptron algorithm might benefit from incorporating the opening used in

[4] For an explanation of back-transforming, see this article from MedCalc.

each match. Another potential feature that might be important for a match might be the playstyle- how aggressive- of the players. Would having a more aggressive playstyle increase a player's chance of winning? While playstyle is not provided by the Lichess API, the API does provide the details for every move made in the game, including the captures. An aggressiveness feature could be engineered based on, for example, the number of captures a player makes within a set number of moves.

Our analyses gave us a great understanding. Although our initial regression attempts and binning strategies provided incredibly high R-squared values, we learned that these high values were not completely reliable. This was because of presumed assumptions and oversimplification. Specifically, fitting a linear regression over the average win percentages of binned rating differentials did not work. We confirmed that rating differentials significantly impact win probabilities and found partial solutions to whether game outcomes are based on player ratings. Also, based on our results, the high R^2 value solely is not a predictor of accuracy. We identified common patterns and realized the need for more refined approaches. Some actions we will take as a result of the analysis would be getting more granular data, adding new predictive features, and looking toward non-linear models/methods. However, at least these early steps helped us get accustomed to the data and gain confidence. Some anticipated questions were how specific openings like the French could significantly impact game length, and future analyses will look carefully at opening strategies and playstyles.

We transitioned to perceptron models for simple 'win versus lose' forecasts. However, we did not get high precision. Still, we learned that chess is complex, so dependency on just rating differences and a few variables is insufficient. We learned to utilize more details like move-by-move details or features representing playstyle could help us in the future. We learned about understanding data and pitfalls. Overall, we learned more features, and we had a great chance of creating more accurate predictions.