Player rating prediction

# Scrabble game overview

**Objective of the Game**

Scrabble is a word game where players use letter tiles to form words on a game board. The objective is to score as many points as possible by creating words with high-value letters and strategic placement on the board.

**Game Setup**

Board: The board is a 15x15 grid, with certain squares offering bonus points, like

Double Word Score, Triple Word Score, Double Letter Score, and Triple Letter Score.

Tiles: Players start with a set of seven letter tiles drawn from a bag. Each tile has a specific point value based on the letter’s frequency and difficulty of use (e.g., "Q" and "Z" are high-value tiles).

**Gameplay**

1. Turns: Players take turns to make a word on the board. A word must connect to existing words on the board (after the first word is placed). Players can form words horizontally or vertically, similar to a crossword puzzle.

2. Word Scoring: Each tile has a point value, and the word score is the sum of these tile points. Placing tiles on bonus squares can multiply the score. Words that connect with other words (forming multiple words in one play) get points for each valid word created.

3. Options on a Turn:

Play : Place tiles on the board to form a word.

Exchange: Swap one or more tiles with new ones from the bag (only allowed if there are enough tiles remaining).

Pass: Skip the turn, which might be strategic if the player has poor tiles or is planning ahead.

4.Challenges: If a player thinks an opponent’s word is invalid, they can challenge it. If the word is not in the agreed dictionary, the move is removed, and the player loses that turn.

**How game ends**

The game ends when either:

- A player has used all their tiles, and no more tiles are left to draw.

- Both players pass consecutively or reach a turn limit in digital games.

The final score is the sum of each player’s points minus the value of any remaining tiles in their rack. The player with the highest score wins.

Key Scoring Strategy

Players aim to maximize their points by:

- Forming high-scoring words using valuable letters.

- Utilizing premium squares effectively.

- Creating multiple words in a single turn by connecting new words with existing ones.

# Dataset overview

**Objective**This project involves predicting the ratings of Scrabble players on Woogles.io based on gameplay data. The main goal is to use metadata and gameplay information to predict the ratings of human opponents in the test dataset (test.csv) using a machine learning model. The evaluation metric for this prediction is Root Mean Squared Error (RMSE).

**Data Description**The dataset contains information on approximately 73,000 Scrabble games played by three bots on Woogles.io—BetterBot (beginner), STEEBot (intermediate), and HastyBot (advanced). These games were played between bots and regular registered users. The primary datasets used for prediction include metadata about the games as well as detailed gameplay data (i.e., individual turns), helping to capture aspects of each game’s dynamics.

The project uses data from multiple files:

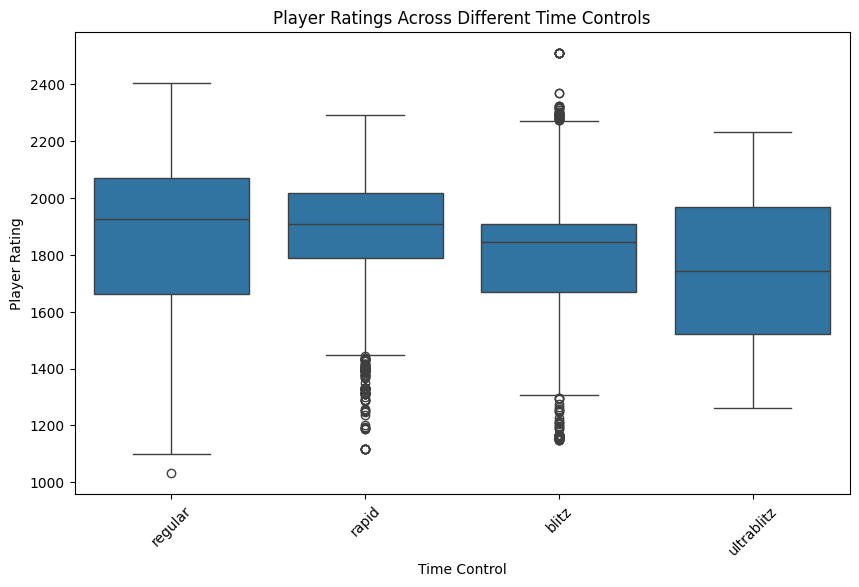
* **games.csv**: Contains metadata for each game, including details like who went first, time controls, and game end conditions.
* **turns.csv**: Contains data on each turn taken during the games, including player racks, moves, locations, and points scored per turn.
* **train.csv and test.csv**: Contain final scores and player ratings for each player in each game, specifically the player’s rating prior to the game. **train.csv** includes this information, whereas **test.csv** has missing ratings for human players, which is the target variable for prediction.
* **sample\_submission.csv**: A reference format for creating prediction submissions.

### Data Files Summary

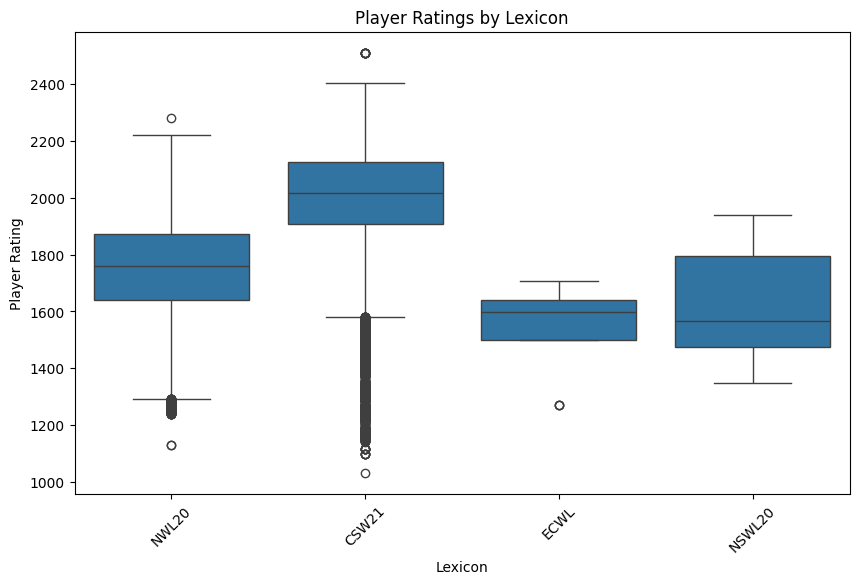
1. **games.csv**Contains game metadata for each game, including:
   * game\_id: Unique identifier for each game
   * first: Indicates which player went first
   * time\_control\_name: Name of the time control used (e.g., "regular", "rapid", or "blitz")
   * game\_end\_reason: Reason the game ended
   * winner: Winner of the game
   * created\_at: Timestamp of game creation
   * lexicon: English lexicon used in the game (e.g., "CSW19", "NWL20", "CSW21")
   * Other time control details
2. **turns.csv**Contains detailed gameplay information for each turn, including:
   * game\_id: Unique identifier linking turns to a specific game
   * turn\_number: Order of the turn within the game
   * nickname: Username of the player making the move
   * rack, location, move, points, score: Detailed play data such as the rack at turn start, move made, points earned, and cumulative score
   * turn\_type: Type of turn (e.g., "Play", "Exchange", "Pass")
3. **train.csv and test.csv**Provide data on the final scores and ratings for each player before each game:
   * game\_id: Unique identifier for each game
   * nickname: Username of the player
   * score: Final score in each game
   * rating: Player’s rating before the game (the target variable to predict in test.csv)
4. **Sample\_submission.csv**A reference file with the correct format for creating the prediction output.

# Checking how different features affects rating(EDA)

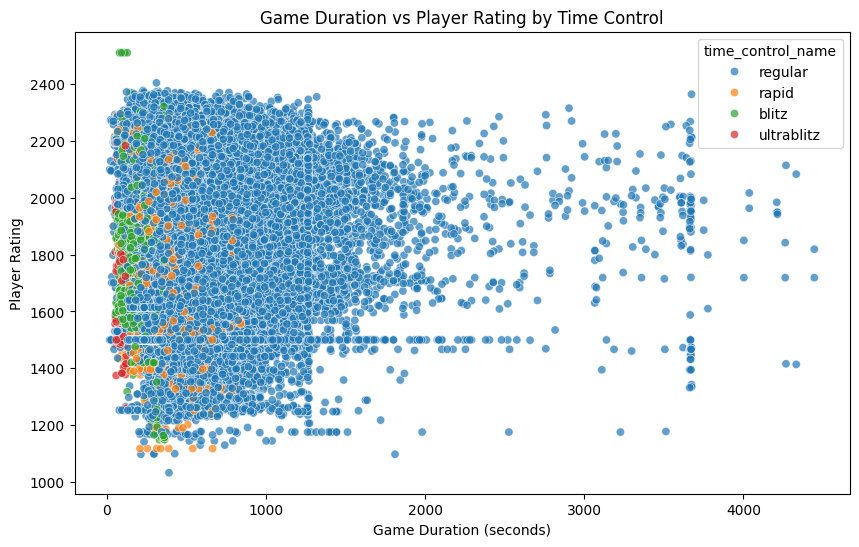
1. **Basic Data Exploration**
   * **Checking Missing Values**: We’re examining each file, especially train.csv and test.csv, for any missing data. This will help us identify any necessary preprocessing steps and ensure that our analysis remains consistent and accurate.
   * **Summary Statistics**: We’re calculating basic statistics (mean, median, min, max, and standard deviation) for numerical columns like scores and points. Understanding these statistics provides insight into score patterns and variability, helping us detect any potential correlations with player ratings.
   * **Distribution of Ratings**: We’re plotting the distribution of ratings in train.csv to understand the spread and symmetry of player ratings. This can reveal whether ratings follow a normal distribution or are skewed, which will help inform our approach to predictive modeling.
2. **Game-Level Analysis (in games.csv)**
   * **Time Control vs. Ratings**: We’re analyzing if different time controls (indicated by time\_control\_name) relate to player ratings. By plotting average player ratings across various time controls, we can observe if certain controls favor higher-rated players or lead to better performance overall.



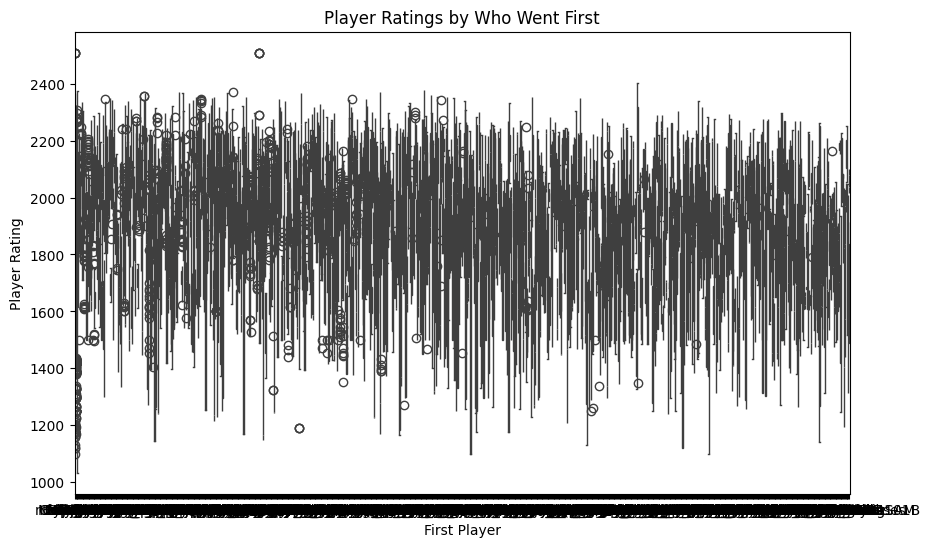
* + **Lexicon Analysis**: We’re examining how player ratings vary across different lexicons (e.g., CSW19, NWL20) by plotting rating distributions for each. Certain lexicons might correlate with higher player ratings, potentially due to different play strategies or familiarity with specific word lists.



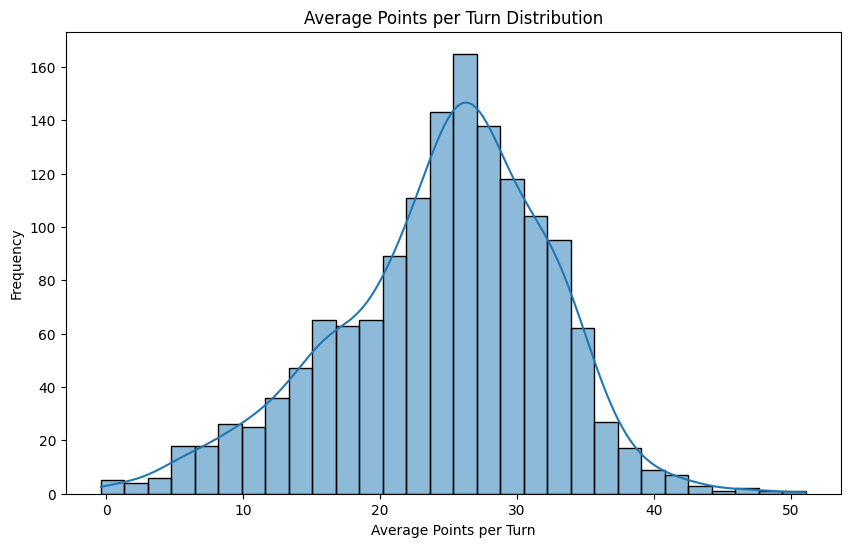
* + **Game Duration**: We’re investigating whether longer game durations (as shown by game\_duration\_seconds) are associated with higher-rated players. This could suggest that skilled players often engage in longer, more strategic games.



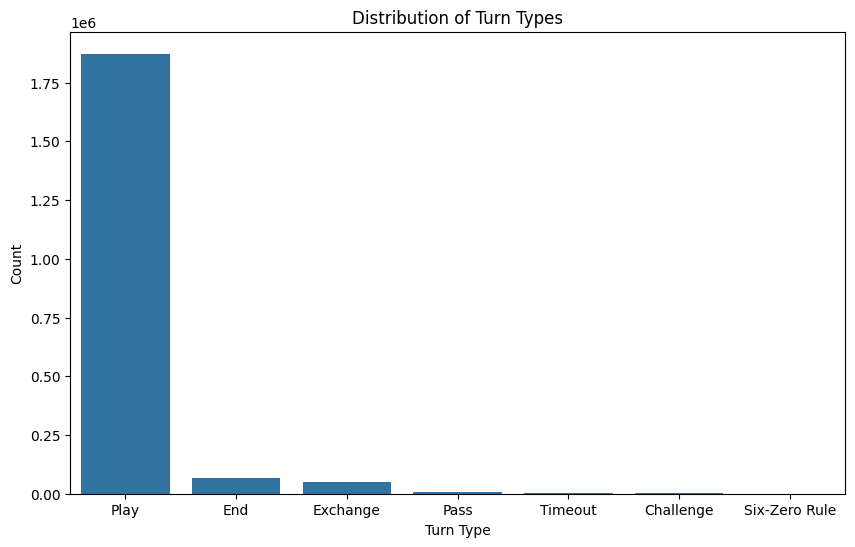
* + **Who Goes First**: By examining the average ratings of winners and losers who go first, we’re exploring whether having the first move correlates with better performance or is advantageous for certain ratings.



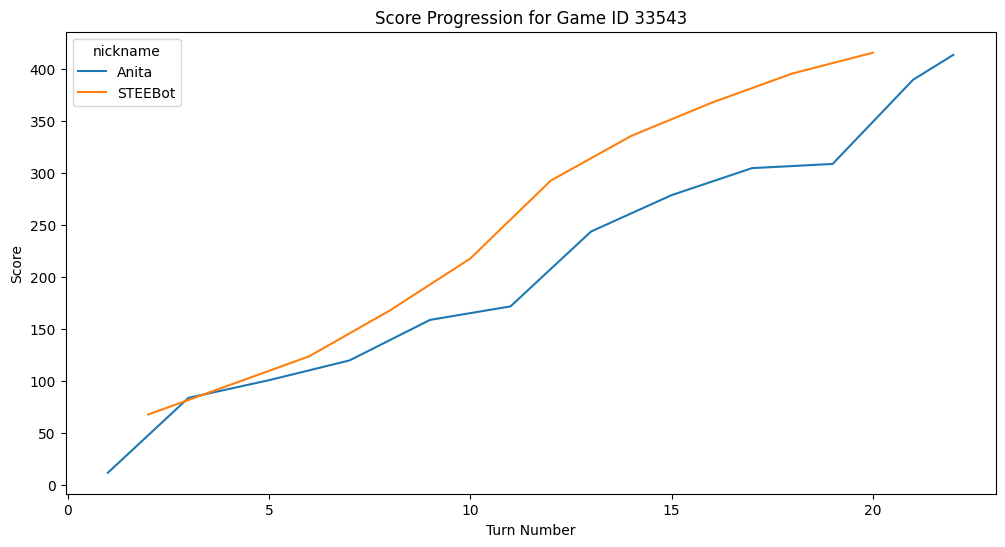
1. **Turn-Level Analysis (in turns.csv)**
   * **Average Points per Turn**: We’re calculating and plotting the average points scored per turn for each player. Higher-rated players may consistently score more points per turn, potentially indicating stronger strategic play and scoring potential.



* + **Turn Types**: We’re analyzing the distribution of different turn types (e.g., Play, Pass, Exchange, Challenge) to see if certain types are favored by higher-rated players. This may show how skilled players use turn types strategically.

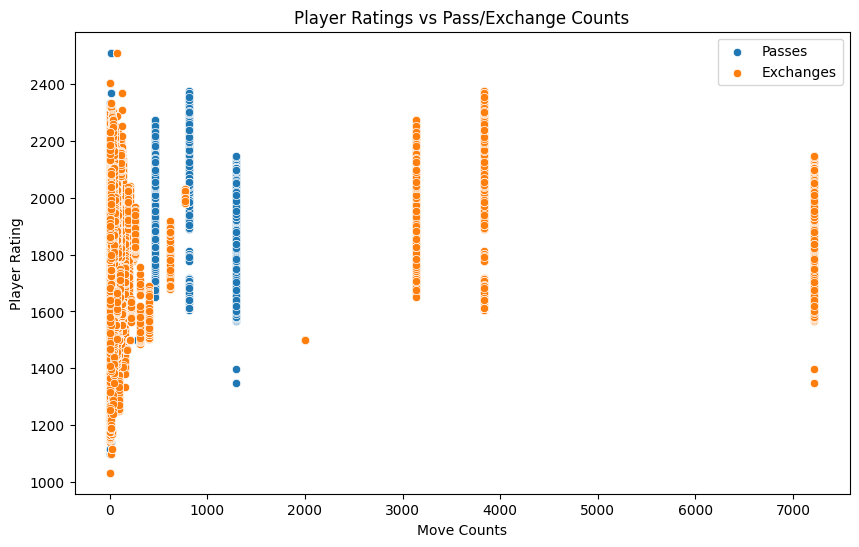


* + **Score Progression**: By plotting score progression turn by turn within each game, we’re examining how score changes reflect player skill. Higher-rated players may maintain a steady lead or score consistently, potentially showing a pattern that correlates with rating.

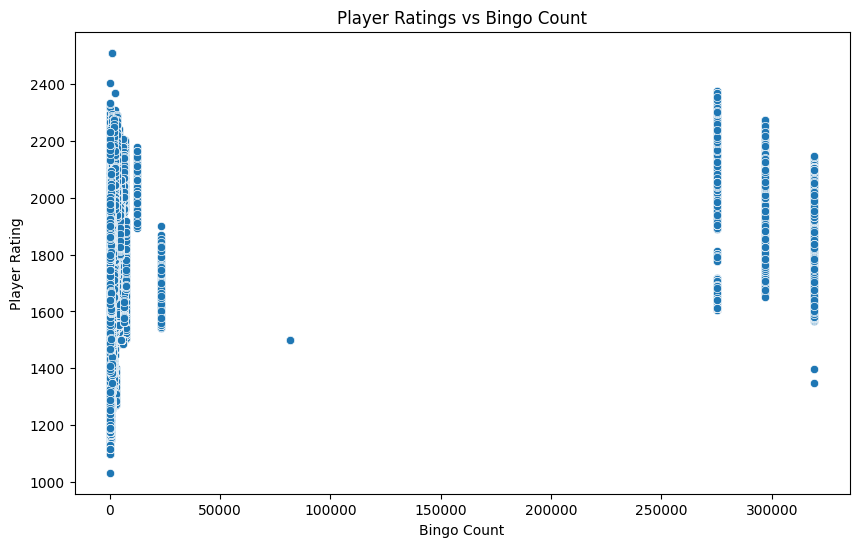


* + **Rack Complexity**: We’re analyzing rack composition by looking at the letters players have each turn, especially high-scoring or blank tiles. Observing how high-rated players manage these racks could reveal effective rack usage as a predictor of success.

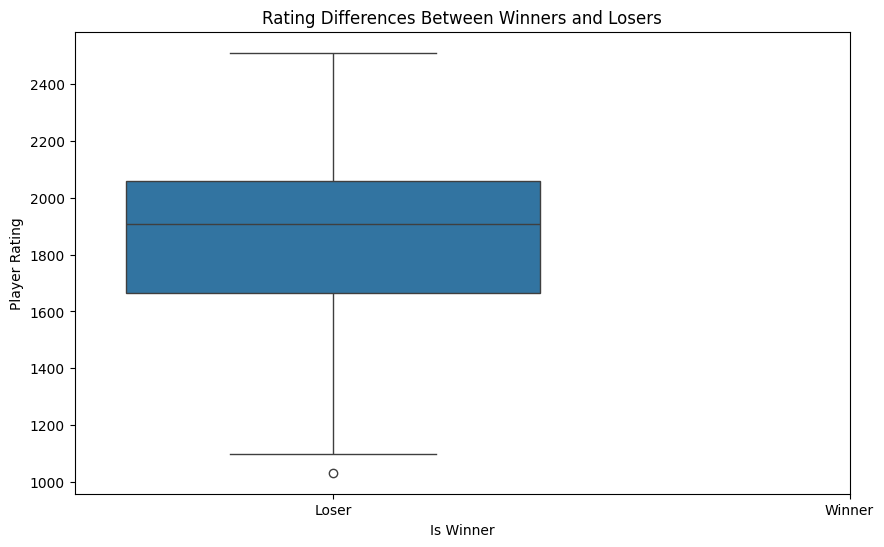
1. **Player Strategy Analysis**
   * **Passes and Exchanges**: We’re exploring the relationship between the frequency of passes or exchanges and player ratings. Higher-rated players may use these moves more deliberately for strategy, while beginners may use them out of necessity.



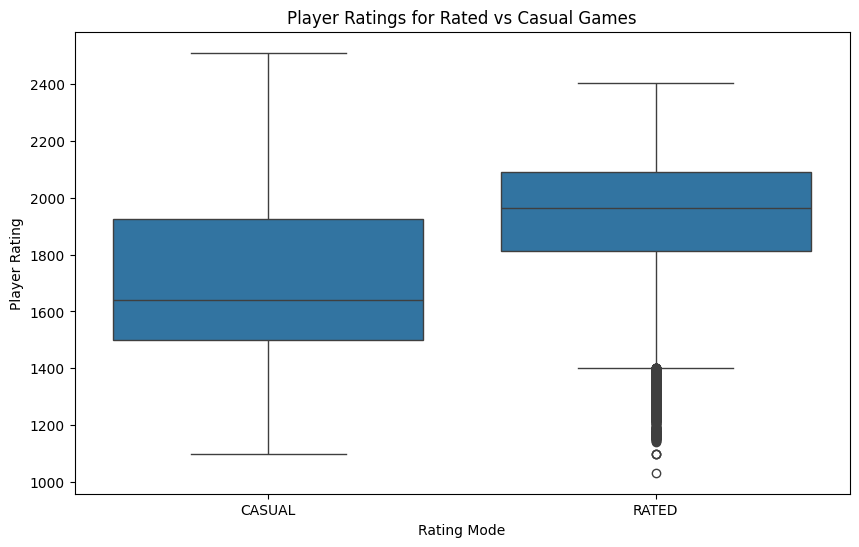
* + **Word Length and Complexity**: In the training data, we’re examining word length and complexity (e.g., usage of less common letters). Higher-rated players might form longer or more complex words, suggesting a stronger vocabulary and scoring potential.
  + **Bingos**: We’re tracking bingo occurrences (when players use all seven tiles) and observing if higher-rated players achieve more bingos. This could indicate that stronger players manage their racks more effectively.



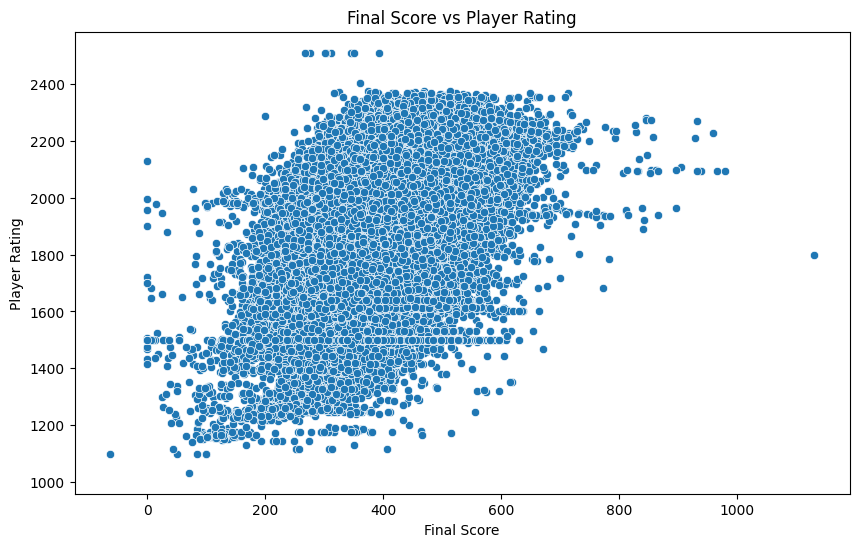
1. **Game Outcomes Analysis**
   * **Winner and Rating Differences**: We’re analyzing the rating differences between winners and losers to see if winning tends to correlate with higher ratings. Additionally, we’re exploring whether different bot levels (BetterBot, STEEBot, HastyBot) make it more challenging for players of various ratings to win.



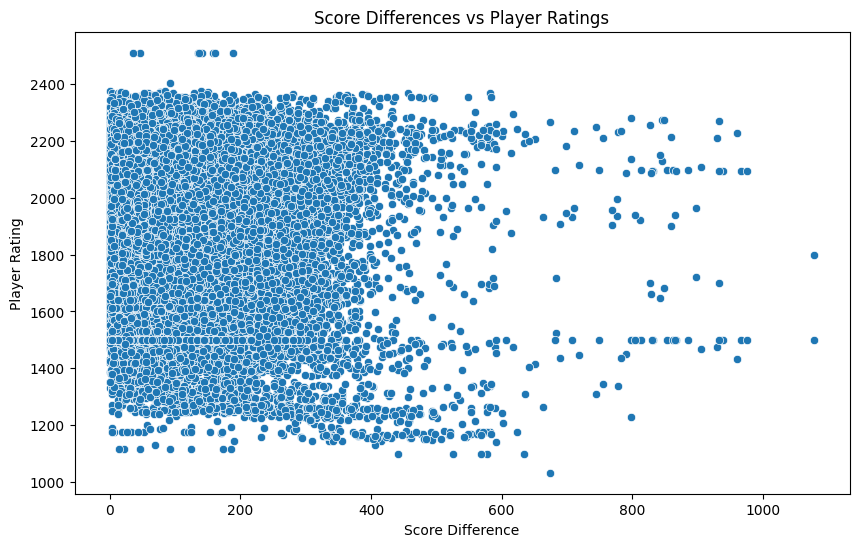
* + **Rating Mode (RATED vs. CASUAL)**: We’re examining if there’s a performance difference between rated and casual games. Players may play differently when a game impacts their rating, which could influence rating distribution.



1. **Relationship between Final Scores and Ratings**
   * **Score vs. Rating Correlation**: We’re plotting the final scores against player ratings in train.csv to assess if higher scores generally indicate higher ratings. A positive correlation here would suggest that scores are a strong predictor of rating.



* + **Score Differences**: By calculating score differences between players and their opponents, we’re checking if high-rated players consistently win by larger margins. This could indicate their strength in dominating games.



1. **Feature Engineering for Model Training**
   * **Average Moves per Game**: We’re calculating the average number of moves per game using the turns dataset. This metric could reflect a player’s pacing and strategy, as players with different ratings may take varying numbers of moves per game.
   * **Average Score per Move**: We’re determining the average score per move from the turns dataset. This feature provides insight into scoring efficiency, as higher-rated players might score more points per turn on average, indicating skillful gameplay.
   * **Game-Level Metadata**: We’re incorporating game-related metadata from the games dataset, such as game\_duration\_seconds and winner. This information provides context on the length of games and game outcomes, which could correlate with player ratings.
   * **Average Points per Turn**: We’re calculating the average points scored per turn to capture consistency in scoring. This feature could indicate a player’s ability to score steadily across turns, which may relate to higher ratings.

# 

# Model training

In this section, I describe the final preparation of the dataset, the evaluation of various regression models, and the process of making predictions based on the trained model.

#### **Data Preparation**

The dataset was divided into training and testing subsets. The training set (train\_df) was created by filtering out entries where the user\_rating was available, while the testing set (test\_df) comprised entries where the user\_rating was missing. The indices of both data frames were reset for clean organization:

*train\_df = main\_df[~main\_df['user\_rating'].isna()].reset\_index(drop=True)*

*test\_df = main\_df[main\_df['user\_rating'].isna()].reset\_index(drop=True)*

#### **Model Definition**

We defined a set of regression models for predicting user ratings. The selected models included:

* **Linear Regression**
* **Decision Tree Regressor**
* **Random Forest Regressor**
* **Neural Network Regressor (MLP)**
* **XG-Boost**
* **KNeighbour-Regression**

This variety of models allows for comparison across different algorithms to identify the best performer for our specific dataset.

#### **Model Evaluation**

To determine which model performs best, I created an evaluation process using cross-validation. This method splits the training data into smaller subsets, allowing each model to be trained and tested multiple times on different parts of the data. During this evaluation, I calculated three important metrics:

1. **R² Score**: This measures how well the model explains the variability of the ratings.
2. **RMSE**: This tells us the average difference between predicted ratings and actual ratings, with lower values indicating better performance.
3. **MAE**: This measures the average magnitude of the errors in predictions, providing a straightforward interpretation of accuracy.
4. **MAPE:** MAPE measures the accuracy of a forecasting method by calculating the percentage error between predicted and actual values.

The process involved:

1. Initializing lists to store scores for each model.
2. Using K-Fold cross-validation to split the training data into training and validation sets multiple times.
3. Training each model on the training subset and predicting on the validation subset.
4. Calculating the evaluation metrics for each fold and averaging them for each model.

The average scores were stored in a dictionary, which was later converted to a DataFrame for easier interpretation:

*score\_summary\_df = evaluate\_regression\_models(regression\_models, X\_train, y\_train, n\_folds=2)*

The results were sorted based on RMSE to identify the best-performing model, which helps in understanding how well each model generalizes to unseen data.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | R2 | RMSE | MAE | MAPE |
| Linear regression | 0.555358 | 153.260169 | 113.565201 | 6.538576 |
| KN-regressor | 0.163680 | 209.218241 | 157.085034 | 8.67602 |
| Decision tree regressor | 0.713658 | 122.816930 | 65.966773 | 3.731457 |
| Random forest | 0.861347 | 85.575324 | 50.230893 | 2.863279 |
| MLP | -0.069038 | 222.584199 | 164.899464 | 9.354445 |
| XG boost | 0.868234 | 83.421700 | 52.575439 | 2.979499 |

#### **Final Model Prediction**

After evaluating the models, I chose the Random Forest Regressor as the best-performing model based on its evaluation metrics. . The model was trained on the X\_train features and y\_train ratings:

*model = RandomForestRegressor()*

*model.fit(X\_train, y\_train)*

Using the trained model, I made predictions for the test\_df, specifically populating the user\_rating field for entries without an existing rating:

*test\_df["user\_rating"] = model.predict(test\_df.drop(["user\_name", "user\_rating"], axis=1))*

#### **Submission File Preparation**

Finally, I prepared the submission file containing the predicted user ratings for the test set. This involved selecting the relevant columns (game\_id and user\_rating), renaming the user\_rating to rating to match the required submission format, and exporting the DataFrame to a CSV file:

*submission.to\_csv("submission.csv", index=False)*

*from google.colab import files*

*files.download("submission.csv")*