Decoding Chess: Analysis of 20,000 games

Devarsh Shah, Abhiram Naredla, Veer Patel

Project Goal

- We analyzed a dataset comprising over 20,000 games from Lichess.org to identify trends, strategies and factors that affect the outcome of a chess game.
- The project aims to gain insights into winning strategies for both the white and black pieces, and investigate the relationship between specific chess openings and game results.
- Audience: casual and beginner level chess players.

Dataset

- We used a dataset from Kaggle.com that comprises over 20,000 rows taken from Lichess.org, the second most popular online chess server.

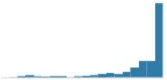
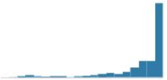
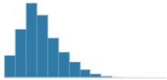
- The dataset has a CC0: Public Domain license meaning we can work on it without needing permission (https://creativecommons.org/publicdomain/zero/1.0/).

- Link to the dataset: https://www.kaggle.com/datasets/datasnaek/chess/data

Raw Data

About this file

20,058 Games from Lichess.org

id	rated	# created_at	# last_move_at	# turns	victory_status	winner	increment_code	white_id
19113 unique values	 true 16.2k 81% false 3903 19%	 1377b 1504b	 1377b 1504b	 1 349	resign 56% mate 32% Other (2586) 13%	white 50% black 45% Other (950) 5%	10+0 38% 15+0 7% Other (11026) 55%	**9438** unique values
TZJHLljE	FALSE	1.50421E+12	1.50421E+12	13	outoftime	white	15+2	bourgris
l1NXvwaE	TRUE	1.50413E+12	1.50413E+12	16	resign	black	5+10	a-00
mIICvQHh	TRUE	1.50413E+12	1.50413E+12	61	mate	white	5+10	ischia
kWKvrqYL	TRUE	1.50411E+12	1.50411E+12	61	mate	white	20+0	daniamurashov
9tXo1AUZ	TRUE	1.50403E+12	1.50403E+12	95	mate	white	30+3	nik221107

Feature Table

Column Name	Rationale
rated	Whether the game involves player rating or not (True or False)
turns	Number of turns in a game.
victory_status	This shows how the game was won (resign, checkmate, or out of time).
winner	This shows who won the game (white or black).
white_rating	This is the rating of the player with white.
black_rating	This is the rating of the player with black.
opening_name	This is the name of the chess opening used

Proposed Method

1. Exploratory Data Analysis
    - Data Inspection
    - Data Cleaning
    - Descriptive Statistics
2. Engineering a Machine learning model
    - Logistic Regression, MLP Classifier, KNN Classifier and Random Forest Classifier
3. Testing our model
4. Data Visualization

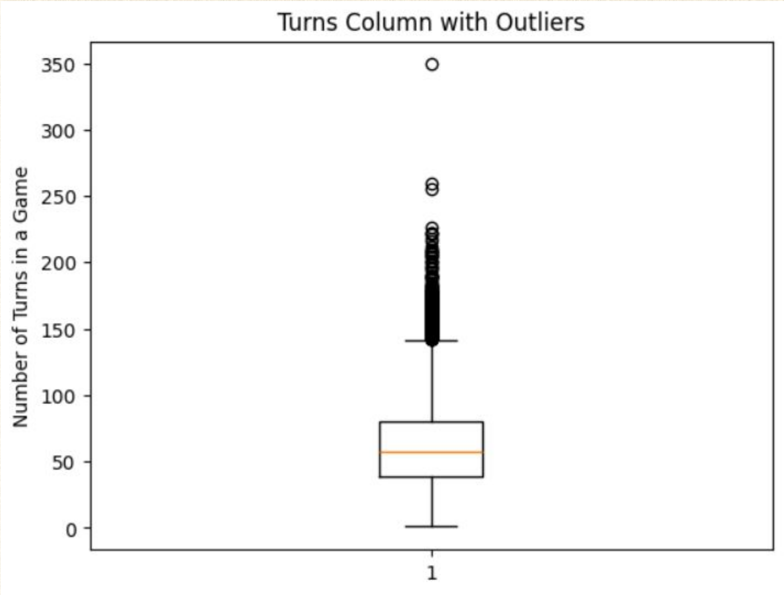
Data Cleaning

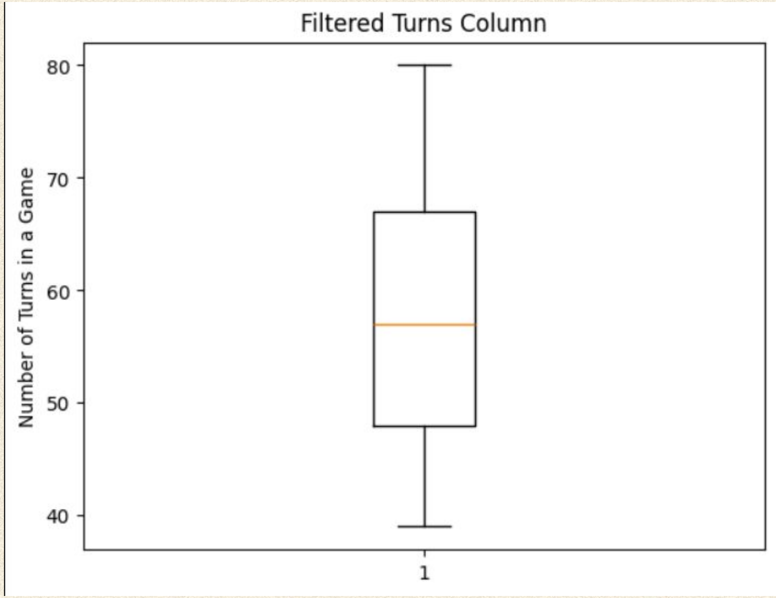
1. "opening_name" column had 1477 different variations. We brought this down to 169 by only accounting for the type of opening and not any variations of the opening. For instance: "Slav Defense: Exchange Variation" will just be Slav Defense in our model.

```
# Clean up the opening_name column to exclude sub-variations of standard chess openings
df['opening_name'] = df['opening_name'].str.split(':').str[0]
df['opening_name'] = df['opening_name'].str.split('#').str[0]
df['opening_name'] = df['opening_name'].str.split('|').str[0]
df['opening_name'].nunique()
```

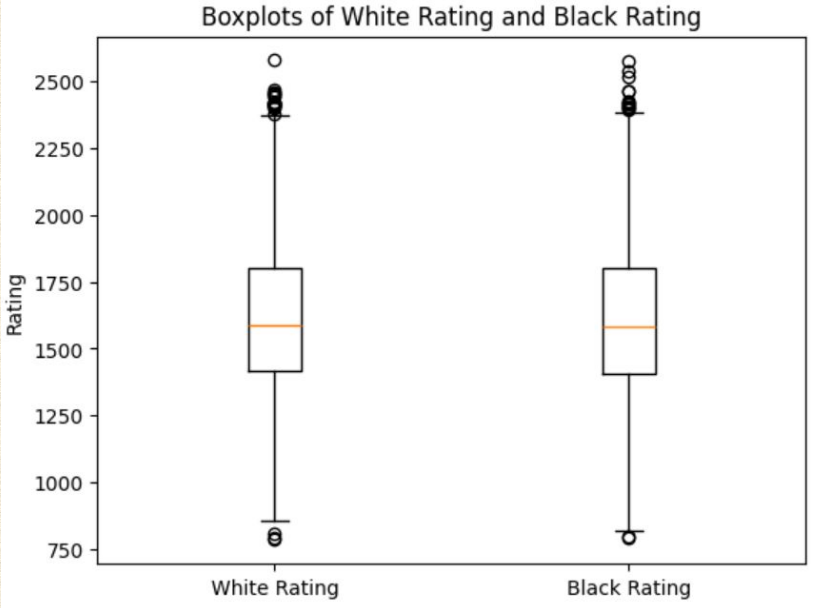
```
169
```

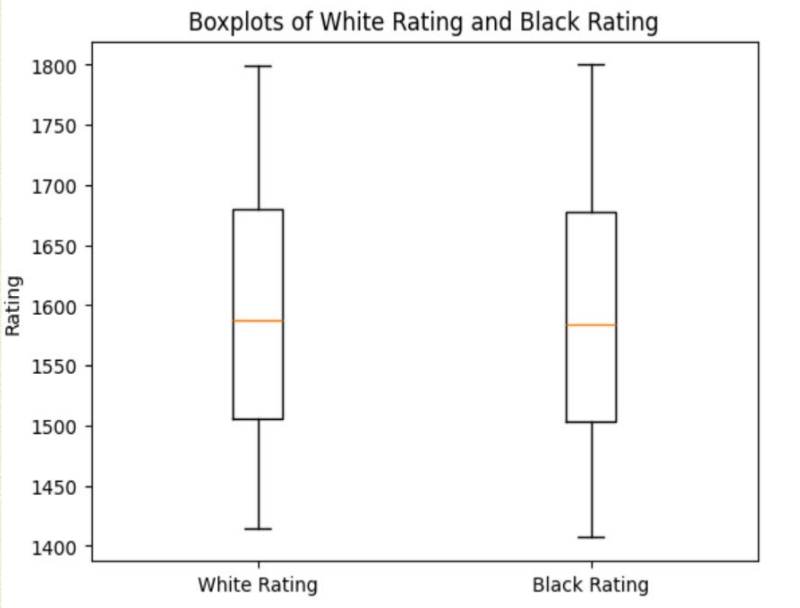
Data Cleaning



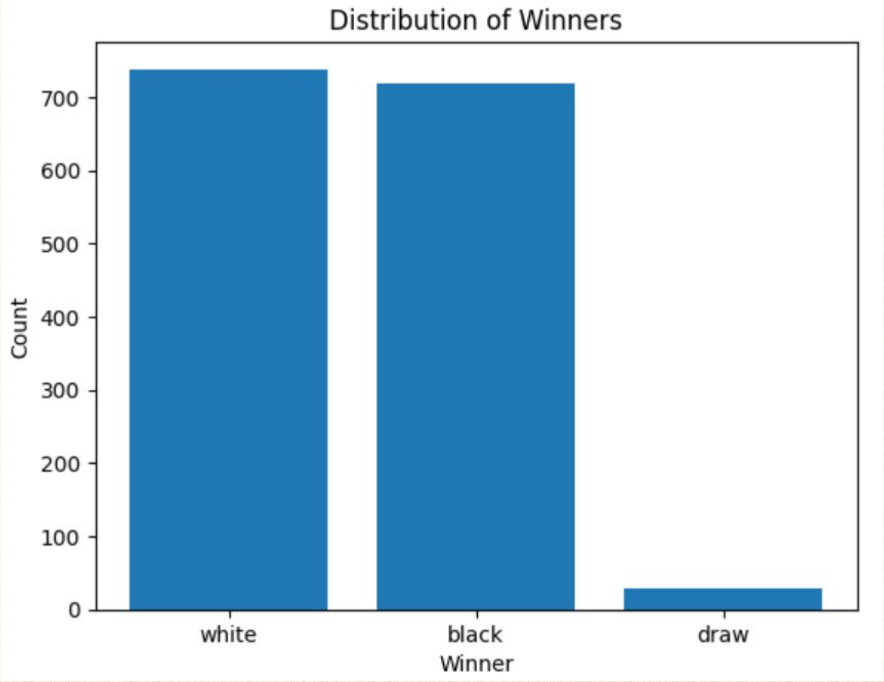


Data Cleaning





Data Cleaning



Since our dataset contains very few “draw” values, we decided to remove them altogether.

Machine Learning Model

- We utilized a 90-10 train-test split for our model.
- We trained 4 different models: Logistic Regression, MLP Classifier, k-Nearest Neighbours Classifier, and Random Forest Classifier.
- We decided on using accuracy as the measurement tool for our model, which is the number of correct predictions in relation to the total number of predictions made by the model.
- Upon testing these models, we found that Logistic Regression gives us the highest accuracy for our data, which is 60%.

Machine Learning Model

```
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

lr_classifier = LogisticRegression(solver='lbfgs',max_iter=10000)
mlp_classifier = MLPClassifier(solver='lbfgs', alpha=1e-5,
                               hidden_layer_sizes=(10, 2), random_state=33,max_iter=10000)
knn_classifier = KNeighborsClassifier(n_neighbors=5)
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
lr_classifier.fit(X_train.to_numpy(),y_train.to_numpy())
```

```
LogisticRegression
LogisticRegression(max_iter=10000)
```

```
mlp_classifier.fit(X_train.to_numpy(),y_train.to_numpy())
```

```
MLPClassifier
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(10, 2), max_iter=10000,
              random_state=33, solver='lbfgs')
```

```
knn_classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

```
rf_classifier.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
from sklearn.metrics import accuracy_score

y_predicted_lr = lr_classifier.predict(X_test)
lr_accuracy_score = accuracy_score(y_predicted_lr,y_test)

y_predicted_mlp = mlp_classifier.predict(X_test)
mlp_accuracy_score = accuracy_score(y_predicted_mlp,y_test)

y_pred = knn_classifier.predict(X_test)
knn_accuracy = accuracy_score(y_test, y_pred)

y_predicted_rf = rf_classifier.predict(X_test)
rf_accuracy_score = accuracy_score(y_predicted_rf, y_test)

print (f"Accuracy of the Logistic Classifier = {lr_accuracy_score}")
print (f"Accuracy of the MLP Classifier = {mlp_accuracy_score}")
print (f"Accuracy of the knn = {knn_accuracy}")
print(f"Accuracy of the Random Forest Classifier = {rf_accuracy_score}")
```

```
Accuracy of the Logistic Classifier = 0.6027397260273972
Accuracy of the MLP Classifier = 0.4931506849315068
Accuracy of the knn = 0.5547945205479452
Accuracy of the Random Forest Classifier = 0.5753424657534246
```

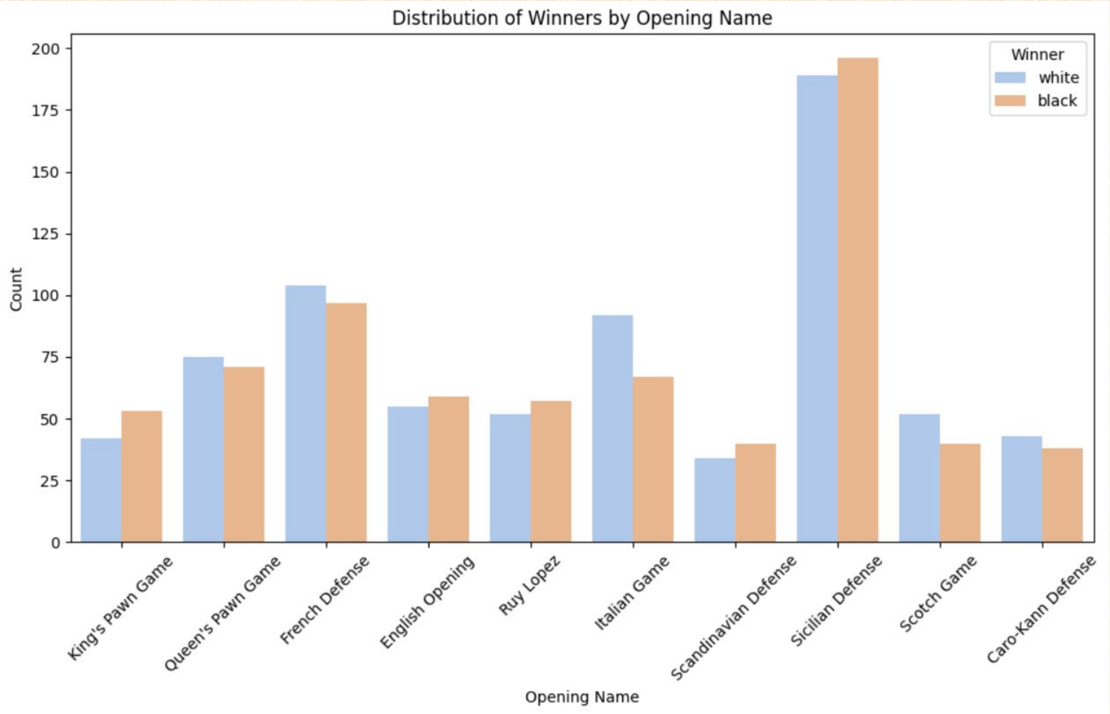
Model Demo

- Input variables: white rating, black rating, number of turns and opening name
- Output: Winner of the game (black or white pieces)
- Target Audience: casual and beginner level chess players.

```
# Sample input data for testing
turns = 13
white_rating = 1500
black_rating = 1191
opening_name = "Slav Defense"

test_data = pd.DataFrame([[turns, white_rating, black_rating, opening_name]], columns=['turns', 'white_rating', 'black_rating', 'opening_name'])

# One-hot encoding to 'opening_name'
test_data_encoded = pd.get_dummies(test_data, columns=['opening_name'], drop_first=True)

missing_columns = set(input_variables_encoded.columns) - set(test_data_encoded.columns)
for col in missing_columns:
    test_data_encoded[col] = 0

# Use classifier to make predictions
y_predicted_lr = lr_classifier.predict(test_data_encoded)
print(f"Predicted Winner: {y_predicted_lr[0]}")
```

```
Predicted Winner: white
```

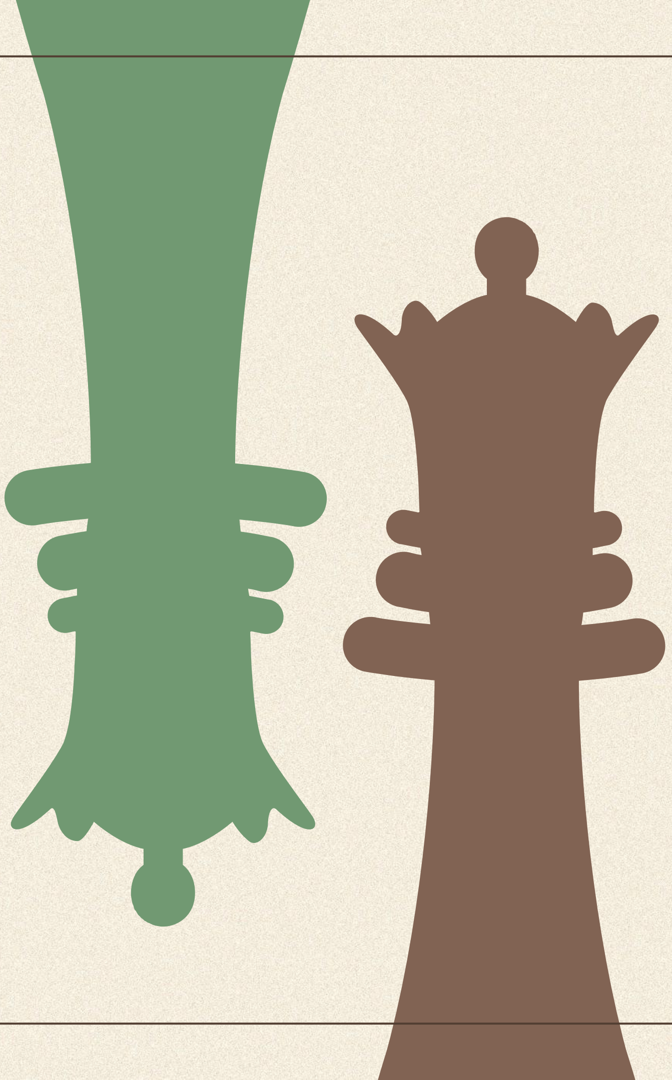
Data Visualization



This bar chart shows the white:black win ratios for each chess opening and shows the better openings for both the white and black pieces.

Result Analysis

- Key Findings:
  - Some chess openings, like the Italian game and Scotch game, have higher win rates using the white pieces and other openings have better win rates using the black pieces.
  - A new chess player could use our findings to select and practice with a specific chess opening to achieve the best win rate.
- Limitations:
  - Initial dataset was only about 20,000 rows and more than 3,000 unique chess openings exist.
  - Certain openings may not have a fair representation in our analysis



Thank You!

Any questions?