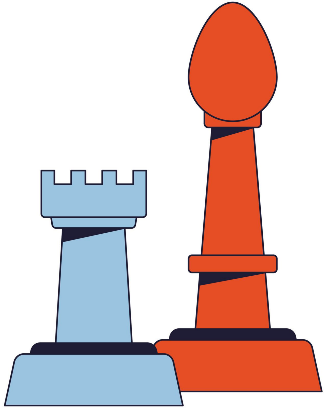
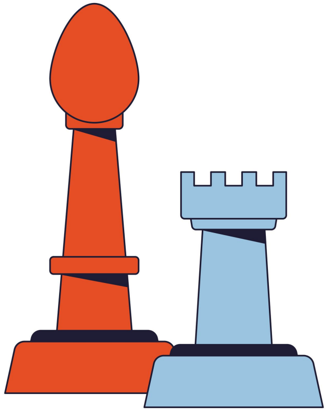


Checkmate



Project 4

Benafsha Alisha, Jonathan Bateson, Ian Denker, Maddie Haughton

Background

Data Set

We looked at a data set showing chess games played in January 2024 on the online site Lichess.

Very large data set with over 130,000 games played.

Used Spark SQL to make data frame using the relevant data.

Next we created the features we were going to use in the prediction models

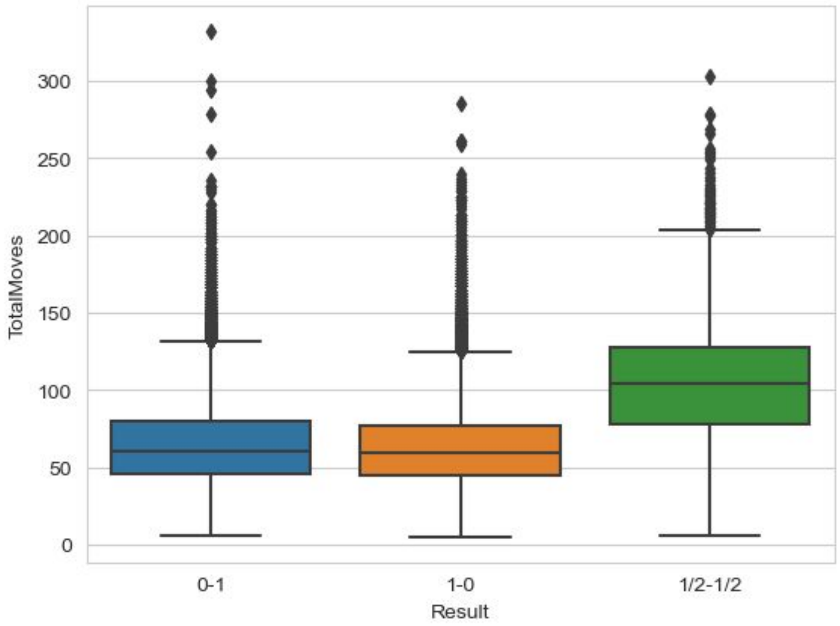
Variables & Setup

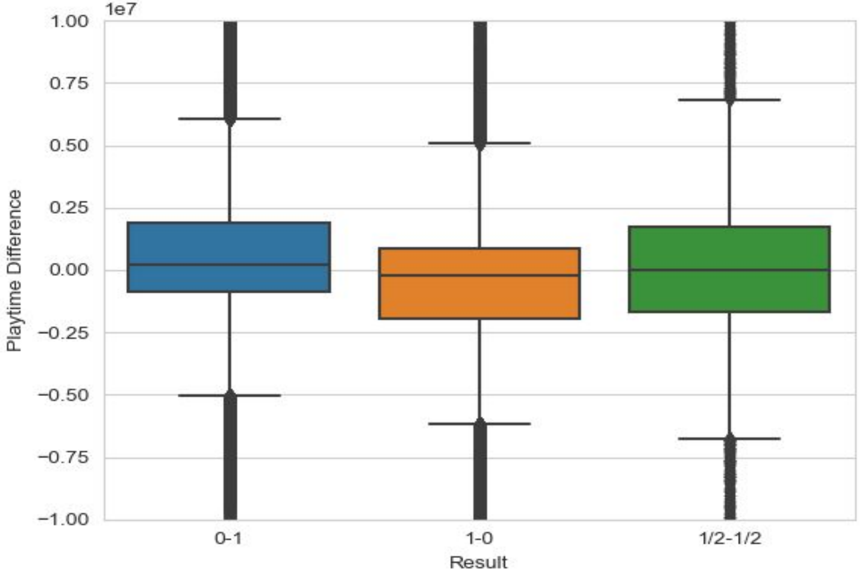
Set up the features we used in the prediction models.

```
# create features exploring the differences in white versus black to use in prediction model
data['Rating Difference'] = data['WhiteElo'] - data['BlackElo']
data['Playtime Difference'] = data['White_playTime_total'] - data['Black_playTime_total']
data['Games Played Difference'] = data['White_count_all'] - data['Black_count_all']
data['White Average Playtime'] = data['White_playTime_total']/data['White_count_all']
data['Black Average Playtime'] = data['Black_playTime_total']/data['Black_count_all']
data['Average Playtime Difference'] = data['White Average Playtime'] - data['Black Average Playtime']
```

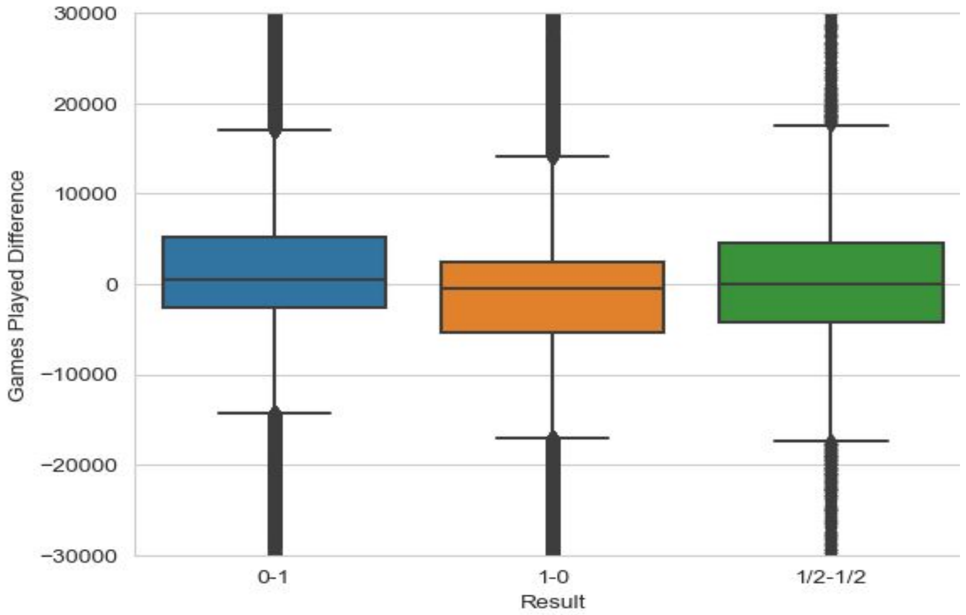
Our first step in using this data was to create box plots for each feature.

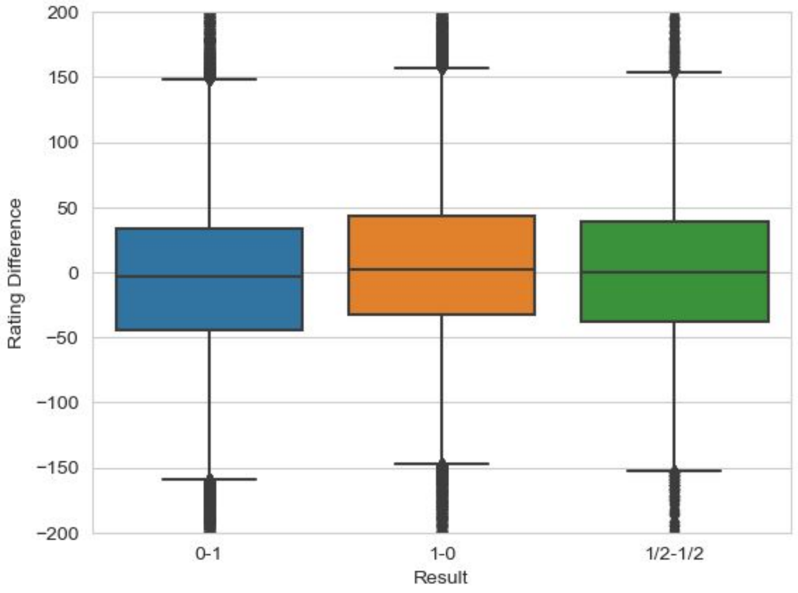
Exploratory Analysis



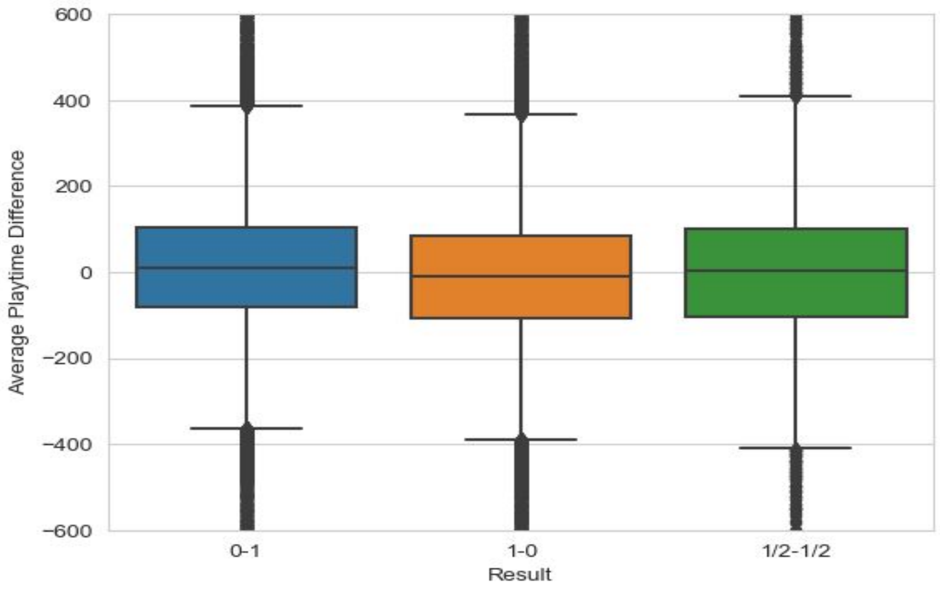


Exploratory Analysis





Exploratory Analysis



Trying Models

Logistic Regression

- Logistic Regression predicts the chance of something happening (like winning or losing) based on past data.
- Unlike linear regression, which predicts continuous values, logistic regression predicts the probability of a binary outcome (0 or 1).
- Regularization techniques like L1 (Lasso) and L2 (Ridge) can be applied to prevent overfitting.
- We check how well the model did by looking at numbers like accuracy and precision.

Logistic Regression

Data Splitting: split into training and testing sets

Model Training: training data with a maximum iteration of 1000.

Outcome:showed precision, recall, and F1-score for each class ('0-1', '1-0', '1/2-1/2') along with support.

The model performed reasonably well in predicting the outcomes, with higher precision and recall for the '1-0' class compared to the other classes.

```
Accuracy Score: 0.5481365965437128
              precision    recall  f1-score   support

         0-1       0.54      0.46      0.50     14516
         1-0       0.56      0.66      0.60     15841
     1/2-1/2       0.30      0.08      0.12      1064

    accuracy                           0.55     31421
   macro avg       0.46      0.40      0.41     31421
weighted avg       0.54      0.55      0.54     31421
```

Random Forest

- Random Forest is a popular machine learning algorithm used for both classification and regression tasks.
- Random Forest builds multiple decision trees.
- It's good at handling lots of different types of data and can make accurate predictions even with missing information.
- Random Forest is a helpful tool in predicting things based on data.

Random Forest

- nitiation: Created a random forest model with 512 trees and a random state of 78.
- Accuracy: Achieved an accuracy score of about 60.16%.
- Report: Generated a report showing precision, recall, and F1-score for each outcome class (Black Wins, White Wins, Draw).
- Feature Importance: Ranked features by importance, with 'TotalMoves' being the most important, followed by 'Playtime Difference', 'Games Played Difference', 'Rating Difference', 'Average Playtime Difference', 'White Average Playtime', and 'Black Average Playtime'.

```
# Displaying results
print(f"Accuracy Score : {acc_score}")
print("Classification Report")
print(classification_report(y_test, predictions, target_names=['Black Wins',
```

```
Accuracy Score : 0.6016040227873078
Classification Report
              precision    recall  f1-score   support

  Black Wins       0.67      0.57      0.62     14516
  White Wins       0.68      0.67      0.67     15841
        Draw       0.00      0.00      0.00      1064

   micro avg       0.67      0.60      0.64     31421
   macro avg       0.45      0.41      0.43     31421
weighted avg       0.65      0.60      0.62     31421
 samples avg       0.60      0.60      0.60     31421
```

```
# Random Forests in sklearn will automatically calculate feature importance
importances = rf_model.feature_importances_
# We can sort the features by their importance
sorted(zip(importances, X.columns), reverse=True)
```

```
[(0.19507156071721687, 'TotalMoves'),
 (0.14279263196713235, 'Playtime Difference'),
 (0.13946888783574055, 'Games Played Difference'),
 (0.132768463350627, 'Rating Difference'),
 (0.13144882239692643, 'Average Playtime Difference'),
 (0.12963849311032632, 'White Average Playtime'),
 (0.12881114062203056, 'Black Average Playtime')]
```

Trying More Models

Neural Networks

- Our model utilized ReLU as a starting point, with 4 total layers and an output layer
- The output layer utilized Softmax with three nodes, as the result column was categorical.
- The loss calculator we used was Categorical Crossentropy, as the output was not binary but ternary and categorical.
- Upon checking with test data, this had a validation accuracy of **0.5697**
- It seems alright at predicting wins/losses, but abysmal at draws.

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_26 (Dense)            (None, 256)               2048

 dense_27 (Dense)            (None, 64)                16448

 dense_28 (Dense)            (None, 64)                4160

 dense_29 (Dense)            (None, 64)                4160

 dense_30 (Dense)            (None, 3)                 195

=================================================================
Total params: 27011 (105.51 KB)
Trainable params: 27011 (105.51 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
Accuracy Score : 0.5697145221348779
Classification Report
              precision    recall  f1-score   support

 Black Wins       0.55      0.60      0.57     14516
 White Wins       0.59      0.58      0.59     15841
       Draw       0.21      0.01      0.01      1064

  micro avg       0.57      0.57      0.57     31421
  macro avg       0.45      0.40      0.39     31421
weighted avg      0.56      0.57      0.56     31421
 samples avg      0.57      0.57      0.57     31421
```

Tuning

- To improve model accuracy, we employed HyperBand tuning.
  - Keeping Softmax as our output, we tried a variety of activation functions on input, selecting from ReLU, tanh, sigmoid, softmax, and swish
  - We also varied the number of layers from 1 to 8, and the number of nodes per layer from 1 to 512
  - Utilizing tuning, we were able to get the model to a validation accuracy of **0.5787**, loss of **0.7691**

```
# Import the kerastuner library
import keras_tuner as kt

tuner = kt.Hyperband(
    create_model,
    objective="val_accuracy",
    max_epochs=20,
    hyperband_iterations=2,
    overwrite=True)
```

```
best_model = tuner.get_best_models()[0]
model_loss, val_accuracy = best_model.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {val_accuracy}")
```

[329]

```
982/982 - 1s - loss: 0.7691 - accuracy: 0.5787 - 802ms/epoch - 817us/step
Loss: 0.769116997718811, Accuracy: 0.57872[illegible]2252616882
```

Binning Outliers

- The last attempt we made that improved accuracy within a neural network model was binning for outliers
- There are many outliers in the data for each feature, but we were most interested in outliers of rating difference
  - These games played where one player is 1500+ ELO rating higher than their opponent surely can’t be representative of most games
  - Placing the rating difference in bins seemed to improve accuracy compared to the original model, but not by much (0.5697 to 0.5715)

```
bins = [-2000, -1000, -500, -150, -50, 0, 50, 150, 500, 1000, 2000]
data['Rating Difference'] = pd.cut(data['Rating Difference'], bins)
```

[160] ✓ 0.0s

```
data['Rating Difference'].head()
```

[161] ✓ 0.0s

```
0      (50, 150]
1    (-150, -50]
2       (-50, 0]
3       (-50, 0]
4       (-50, 0]
Name: Rating Difference, dtype: category
Categories (10, interval[int64, right]): [(-2000, -1000] < (-1000, -500] < (-500, -150] < (-150, -50] ... (50, 150] < (150, 500] < (500, 1000] < (1000, 2000]]
```

+ Code + Markdown

Accuracy Score : 0.5714967696763311

Classification Report

	precision	recall	f1-score	support
Black Wins	0.57	0.50	0.53	14516
White Wins	0.58	0.67	0.62	15841
Draw	0.19	0.05	0.08	1064
micro avg	0.57	0.57	0.57	31421
macro avg	0.45	0.41	0.41	31421
weighted avg	0.56	0.57	0.56	31421
samples avg	0.57	0.57	0.57	31421

Most Successful Model

XGBoost

- Outperformed every other model
- Ensemble of Weak Learner Decision Trees
- Builds one tree at a time, unlike Random Forest
- Gradient Boosting

dmlc XGBoost

```python
from xgboost import XGBClassifier
xg_nn = XGBClassifier(n_estimators = 256, max_depth = 32, objective ='multi:softmax')
xg_nn.fit(X_train_scaled,y_train)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=32, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=256, n_jobs=None,
              num_parallel_tree=None, objective='multi:softmax', ...)
```

```python
predictions = xg_nn.predict(X_test_scaled)
```

```python
accuracy_score(y_test, predictions)
```

```
0.902358295407
```

Does Best Accuracy = Best Model?

Highest score by far

But look at the confusion matrix...

	precision	recall	f1-score	support
Black Wins	0.90	0.93	0.92	14516
White Wins	0.91	0.93	0.92	15841
Draw	0.20	0.04	0.06	1064
accuracy			0.90	31421
macro avg	0.67	0.63	0.63	31421
weighted avg	0.88	0.90	0.89	31421

Resample Model

Upsampled so that draws were more comparable

This model will be correct more consistently across outcomes

	precision	recall	f1-score	support
Black Wins	0.83	0.79	0.81	15675
White Wins	0.82	0.82	0.82	15739
Draw	0.79	0.84	0.81	15746
accuracy			0.81	47160
macro avg	0.82	0.81	0.81	47160
weighted avg	0.82	0.81	0.81	47160

Purely Predictive Model

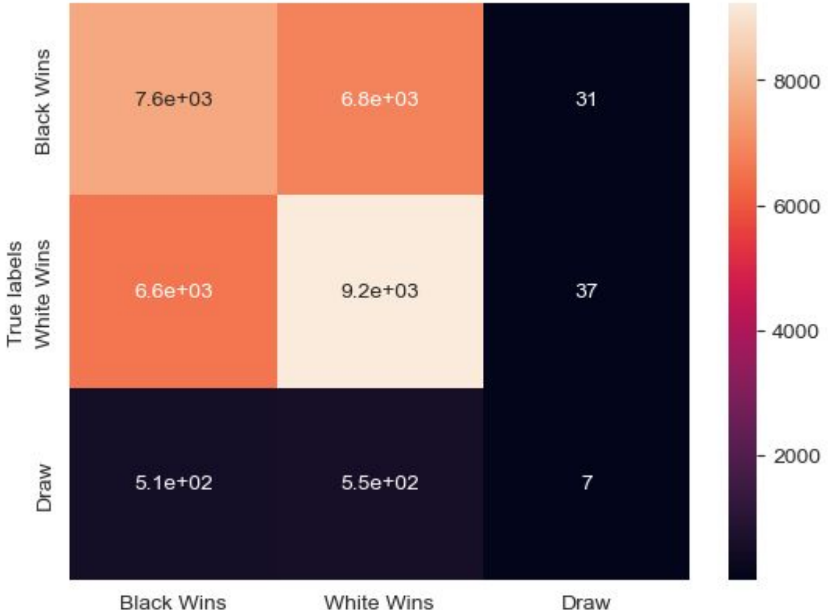
Total Moves is the most important feature (we know that from looking up feature importances earlier)

	precision	recall	f1-score	support
Black Wins	0.54	0.54	0.54	14516
White Wins	0.57	0.61	0.59	15841
Draw	0.08	0.00	0.00	1064
accuracy			0.56	31421
macro avg	0.40	0.38	0.38	31421
weighted avg	0.54	0.56	0.55	31421

But we don’t know that until the game is over.

Can we predict based on other inputs alone?

Not Accurately!



Resampled Predictive Model

Better but still not nearly as good

If we value accuracy the most, it looks like we have to include total moves

	precision	recall	f1-score	support
Black Wins	0.56	0.53	0.54	15675
White Wins	0.57	0.61	0.59	15739
Draw	0.94	0.93	0.93	15746
accuracy			0.69	47160
macro avg	0.69	0.69	0.69	47160
weighted avg	0.69	0.69	0.69	47160

Highest Accuracy We Got

Added More Features

Binned ELO Difference

Resampled

It gets draws up and has an accuracy of 90.9%, even with resampling, so we called it a day with this model

	precision	recall	f1-score	support
Black Wins	0.89	0.88	0.88	15675
White Wins	0.89	0.91	0.90	15739
Draw	0.95	0.93	0.94	15746
accuracy			0.91	47160
macro avg	0.91	0.91	0.91	47160
weighted avg	0.91	0.91	0.91	47160

Let's Test it Out!

The End