Oracle Arena: Predicting NBA Games

Triton Eden and Ryan Peruski

Overview

- Intro
- Data Collection
- Machine Learning
- Results
- Discussion
- Conclusion
- Future Works

ORACLE ARENA

1. Intro

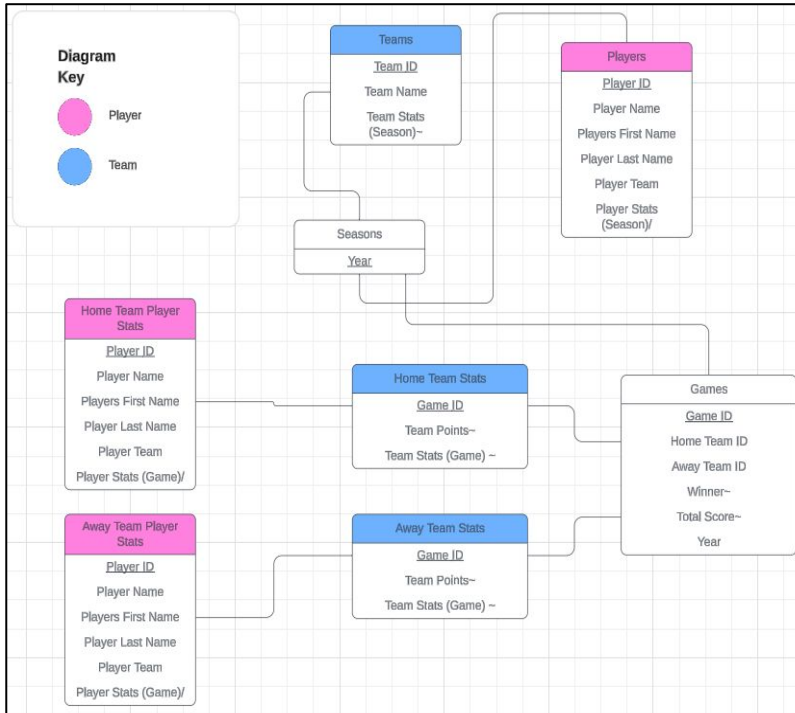
- This project is a subset of a bigger project, Oracle Arena, focusing specifically on the Data Collection and the Machine Learning models
- Predicting winner and total score of NBA games

ORACLE ARENA

2. Data Collection

- PostgreSQL database (Azure)
- NBA API to extract data (harder than it sounds)
- Contains players, teams, stats, and games, 2018-now

SQLAlchemy



3. The NBA API

- Contains stats for past games, live data for present games, and enough game info for future games to warrant a prediction!
- Lots of error handling needed!

```
boxscore = fetch_with_retry(endpoints.boxscoretraditionalv2.BoxScoreTraditionalV2, game_id=game_id)

if boxscore is None:
    print(f"Error fetching stats for game {game_id}: 101. Skipping.")
    continue

try:
    player_stats = boxscore.get_data_frames()[0]
    team_stats = boxscore.get_data_frames()[1]
```

nba_api

An API Client Package to Access the APIs of NBA.com

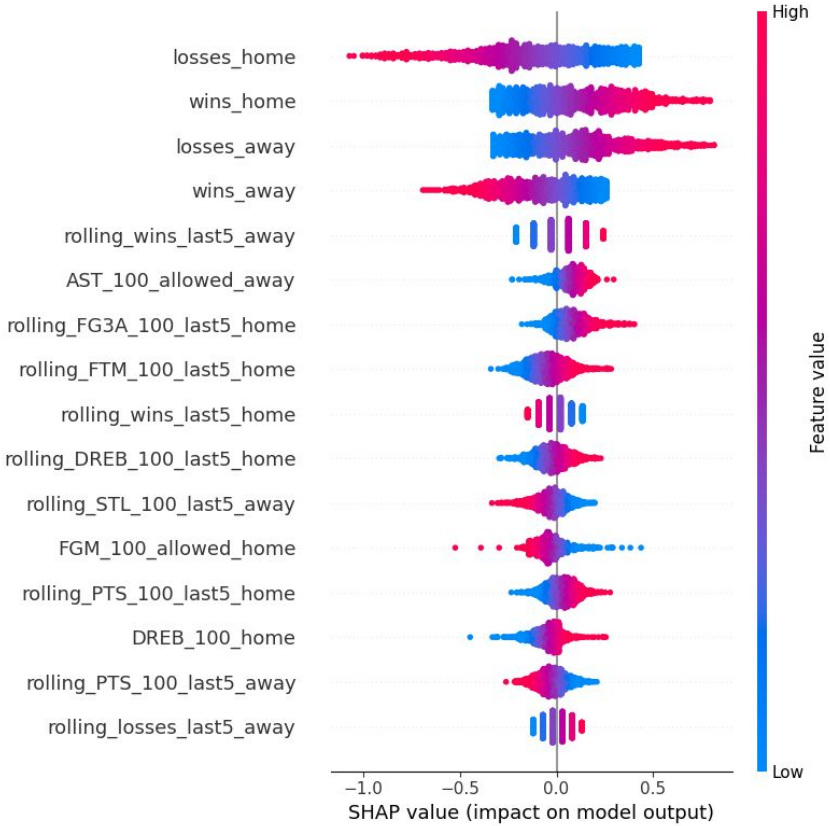
`nba_api` is an API Client for `www.nba.com`. This package intends to make the APIs of NBA.com easily accessible and provide extensive documentation about them.

4. Model Features

Stats Columns:

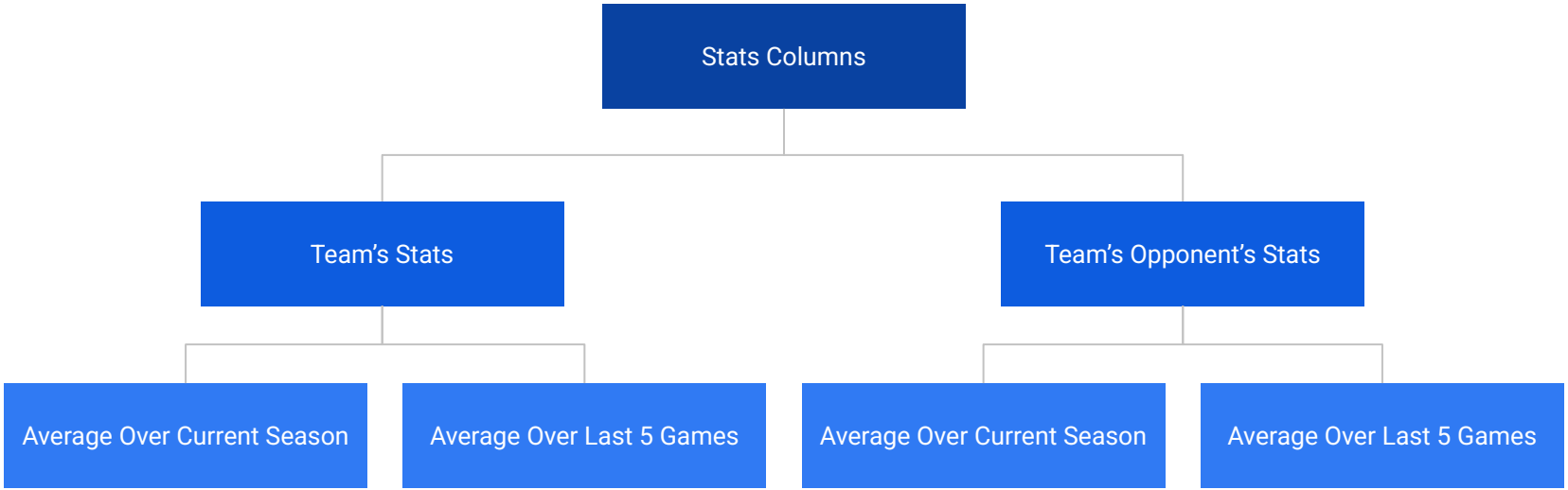
- 2018-19 Season - Now
- FGM, FGA, FG3M, FG3A, FTM, FTA, OREB, DREB, AST, STL, BLK, TO, PTS, POSS, wins, losses
- Besides possessions, wins, and losses, each feature is measured per 100 possessions

- Scaled with Min-Max scaler
- Split into 80% train data and 20% test data
- No shuffle to avoid data leakage



This table is calculated for both teams in each game

5. Model Features



6. Results

Win Prediction			Total Score Prediction		
	Regular Season	Playoffs		Regular Season	Playoffs
	Deep Feedforward Neural Network	Deep Feedforward Neural Network		Model Tuned XGBoost Regression	Ridge Regression
Accuracy	0.66	0.67	RMSE	18.60	16.67
F1 Score	0.71	0.73	MSE	346.33	277.83
Loss	0.62	0.66	R^2 Score	0.11	0.08

7. Models Used - Win Prediction

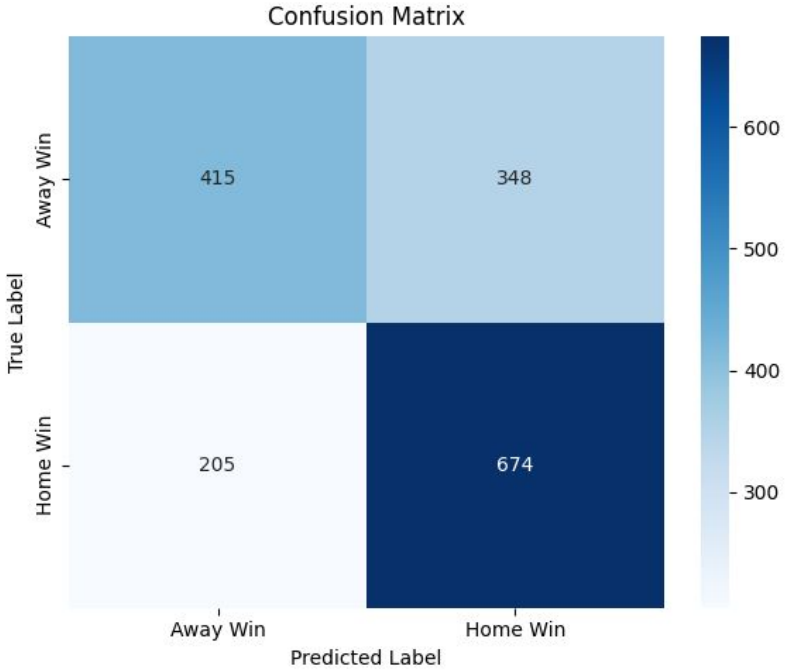
```python
# Deep Feedforward Neural Network

early_stopping = EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6)

model = Sequential()

# First hidden layer
model.add(Dense(256, input_dim=X_train.shape[1], activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.3))

# Second hidden layer
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Third hidden layer
model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Fourth hidden layer
model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.6))

# Output layer
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=Nadam(learning_rate=1e-3), loss='binary_crossentropy', metrics=['accuracy'])
```

Regular Season

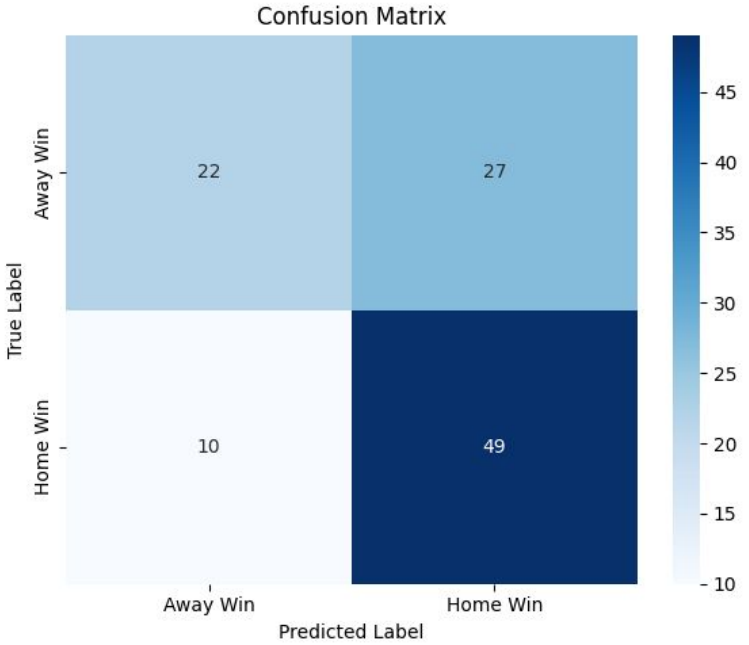
```python
# Deep Feedforward Neural Network

early_stopping = EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6)

model = Sequential()

# First hidden layer
model.add(Dense(256, input_dim=X_train.shape[1], activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))

# Second hidden layer
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))

# Third hidden layer
model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.3))

# Fourth hidden layer
model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.3))

# Output layer
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=Nadam(learning_rate=1e-3), loss='binary_crossentropy', metrics=['accuracy'])
```

Playoffs

8. Models Used - Win Prediction



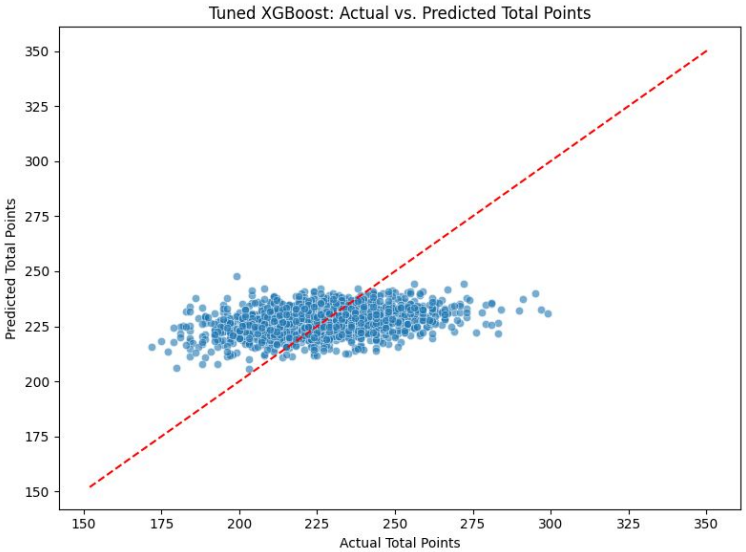
Regular Season



Playoffs

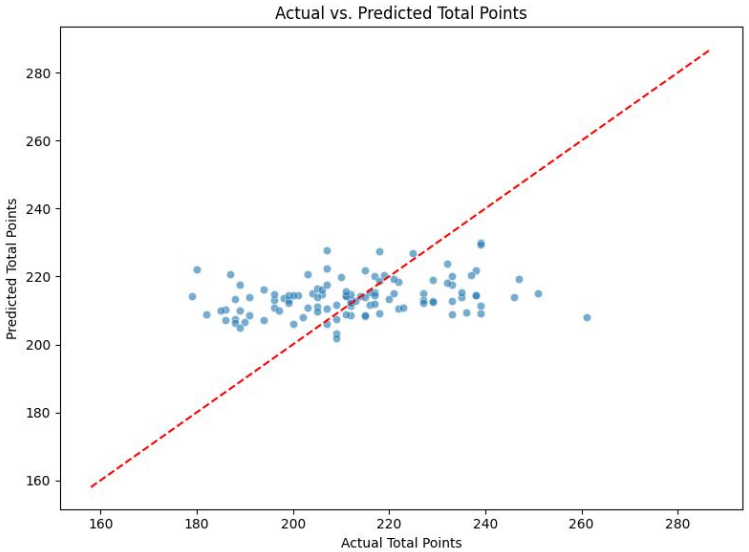
9. Models Used - Total Score Prediction

XGBoost Regressor



Regular Season

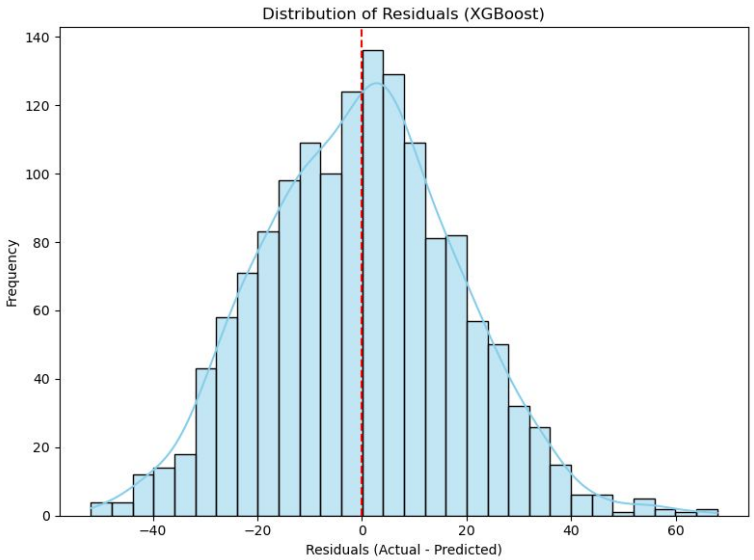
Ridge Regression



Playoffs

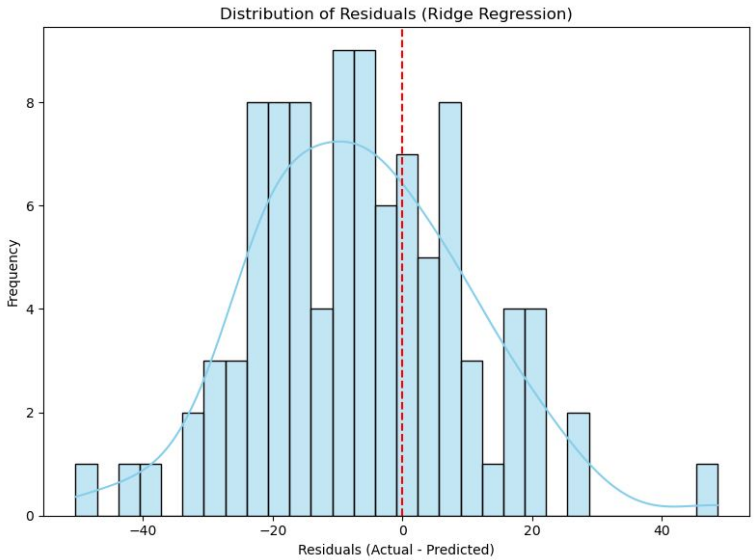
10. Models Used - Total Score Prediction

XGBoost Regressor



Regular Season

Ridge Regression



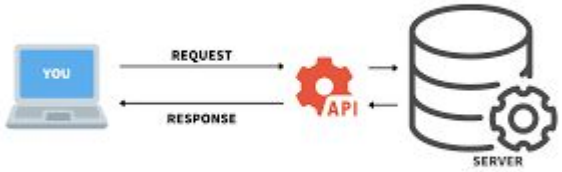
Playoffs

11. Conclusion

- Whenever data is pulled from an external source, it is best to incrementally test and test often. Using API's are harder than they look! Lots of data + multiple languages = lots of cleaning and normalization
- Depending on the size and contents of your training data, model selection and hyperparameter tuning can look very different even for the exact same prediction.
- If a person used our model results as a tool along with their personal intuition, that person WOULD likely successfully gain money in the long run

12. Future Works

- Make our own, more robust, API, that pulls from the NBA stats page, so that we can remove our reliance on the given nba api.
- Test more models and more techniques
    - Uses confidence interval to get better results on a smaller sample set of games



ORACLE ARENA

13. Citations

[1] Swar. (n.d.). *nba_api: An API client for NBA.com*. GitHub. Retrieved April 25, 2025, from https://github.com/swar/nba_api
[2] National Basketball Association. (n.d.). *NBA.com/stats*. Retrieved April 25, 2025, from https://www.nba.com/stats
[3]Microsoft Corporation. (n.d.). *Microsoft Azure*. Retrieved April 25, 2025, from https://azure.microsoft.com
[4]TritonEden. (2025). *Oracle-Arena: Senior Design Project* [Computer software]. GitHub. https://github.com/TritonEden/Oracle-Arena