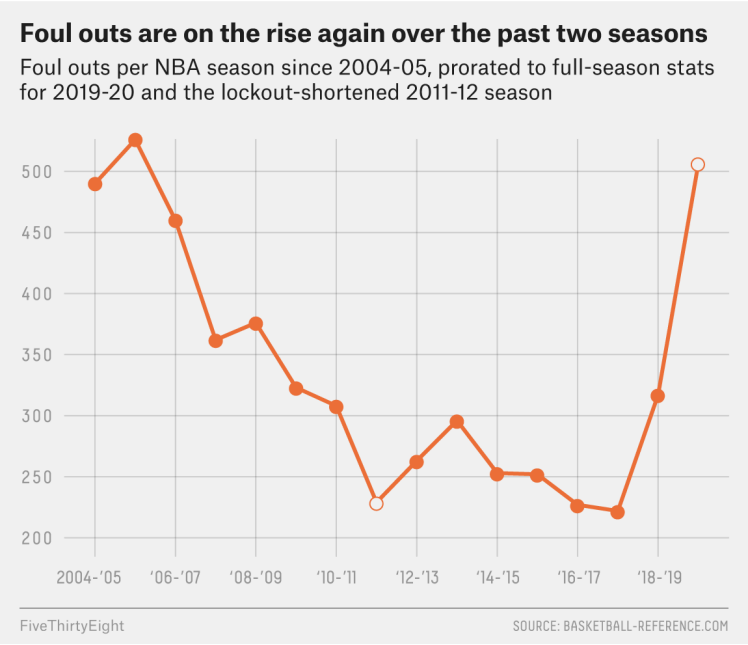
The Foul Factor: Predicting the Probability of Fouling Out in Basketball

Group 5: Karim, Yujin, Alex

1. Introduction

In basketball, a personal foul occurs when a player engages in illegal physical contact with an opponent, such as holding, pushing, or blocking. Committing a foul can lead to several consequences on both the team and player levels. Accumulating team fouls can lead to the opposing team entering the "bonus", granting them free throws or even throw-ins in dangerous areas. Along with this, individual players who accumulate too many personal fouls risk fouling out from the game.

In the NBA, fouling out occurs when a player commits 6 personal fouls and is ejected from the game. Fouling out is seen as a significant concern for coaches and teams due to many reasons. If a key player fouls out, then the team would be forced to play with a less skilled line-up and potentially lose a crucial player during critical moments in the game where individual skill could lead to victory. Along with this, a player fouling out could have a detrimental impact on game flow as coaches would be forced to make unwanted changes to their lineup and tactics.



Research shows that foul-outs have been on the rise in recent seasons which highlights the need to address this issue and utilize resources to tackle it. If teams and coaches can effectively predict scenarios or games when a player is about to foul out, they can make better-informed substitution decisions and gameplan adjustments. The goal of this study is to predict the probability of fouling out using NBA box score data, providing insights into fouling behavior to support strategic decision-making.

2. Data

The dataset used in this study includes NBA game data from 2004 to 2020, sourced from Kaggle. Key variables include box score statistics such as minutes played (MIN), points scored (PTS), assists (AST), rebounds (REB), turnovers (TO), and personal fouls (PF). These variables offer a comprehensive view of player performance and fouling behavior, serving as the foundation for this analysis.

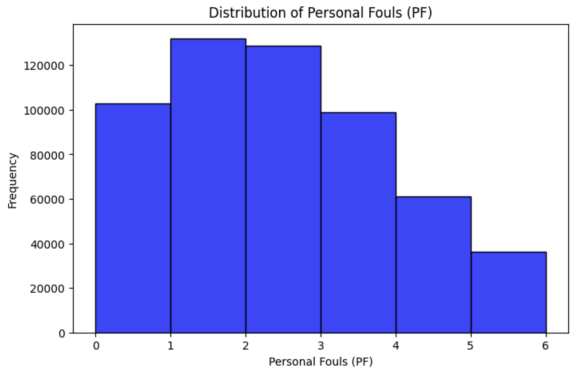
To prepare the data, game details (games_details.csv) were merged with game information (games.csv) using the GAME_ID attribute. Missing values (NA/NaN) were replaced with zeros, and rows with zero minutes played (MIN = 0) were filtered out to focus only on active players. Additionally, a binary target variable, Fouled_Out, was created, where a value of 1 indicates six or more fouls in a game. Rolling averages over the past three games (e.g., MIN_avg_3 and OREB_avg_3) were computed to incorporate historical context into the analysis.

Despite the dataset's depth, it has some limitations. The absence of spatial data restricts the ability to analyze player movements and game dynamics effectively. Furthermore, the data ends in 2020, excluding recent trends and developments in gameplay. Additionally, it does not account for external factors such as player injuries, referee decisions, or in-game contexts, which may significantly influence fouling behavior. While the dataset provides valuable insights, these limitations underscore opportunities for future research and dataset enhancements.

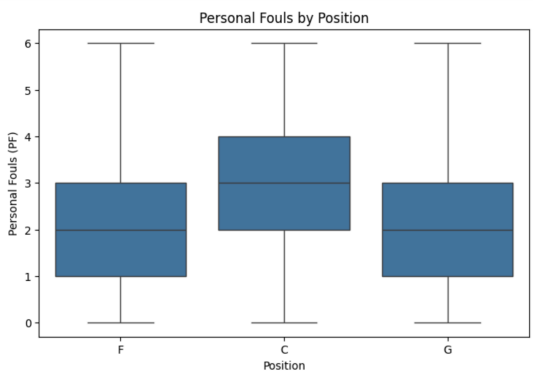
```python
# List of player-level variables
player_vars = [
    'MIN', 'FGM', 'FGA', 'FG_PCT', 'FG3M', 'FG3A', 'FG3_PCT',
    'FTM', 'FTA', 'FT_PCT', 'OREB', 'DREB', 'REB', 'AST', 'STL',
    'BLK', 'TO', 'PF', 'PTS', 'PLUS_MINUS'
]

# Loop through variables and calculate rolling averages
for var in player_vars:
    col_name = f"{var}_avg_3"
    merged_w_noNA[col_name] = (
        merged_w_noNA.groupby('PLAYER_ID')[var]
        .transform(lambda x: x.shift(1).rolling(3, min_periods=1).mean())
    )

```

3. Exploratory Analysis

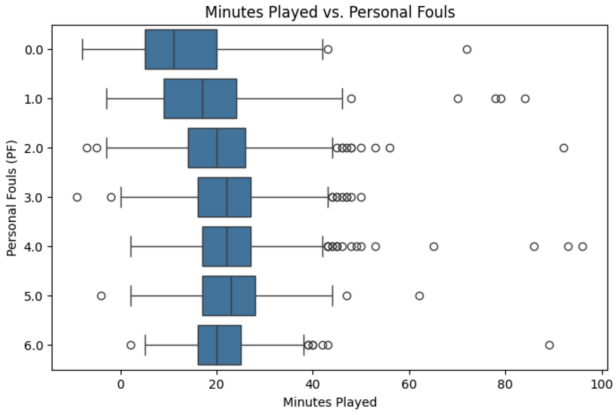
Exploratory analysis was conducted in the form of preliminary visualizations to examine relationships between key player statistics and fouling behavior and help tell the story of the dataset.



As seen in the figure above displaying the frequency distribution of personal fouls, most players in the dataset commit 3 or less fouls per game, yet there are still a good amount of players who commit 4 or more. This shows that there is still a margin of players who foul out or are close to fouling out almost every game which should be addressed by coaches.



In the box plot above, displaying personal fouls by position, it is evident that centers exhibit the highest median of fouls relative to guards and forwards with a range of 2-4 personal fouls per game, while guards commit considerably less fouls per game. This is to be expected due to the nature of each position.



In the box and whisker plot above, the relationship between minutes played and personal fouls committed was studied further. The plot highlights a generally positive trend between fouls and playing time, reinforcing the idea that spending more time on the court is often tied to a higher number of personal fouls committed. Outliers with high playing time and low fouls committed ,may represent exceptionally disciplined players or elite defenders.

The total number of personal fouls (PF) in the dataset was 446 out of 105,622 records, accounting for only 0.42% of the data. This highlights the significant class imbalance within the dataset. Recognizing this imbalance was crucial for fine-tuning the model's hyperparameters and ensuring better performance in handling skewed data.

```python
# determine how many rows in merged_w_3 have PF values of 6 or more, and the percentage and the total number of rows

# Calculate the number of rows with PF values of 6 or more
num_pf_6_or_more = len(merged_w_3[merged_w_3['PF'] >= 6])

# Calculate the total number of rows
total_rows = len(merged_w_3)

# Calculate the percentage
percentage_pf_6_or_more = (num_pf_6_or_more / total_rows) * 100

# Print the results
print(f"Number of rows with PF values of 6 or more: {num_pf_6_or_more}")
print(f"Percentage of rows with PF values of 6 or more: {percentage_pf_6_or_more:.2f}%")
print(f"Total number of rows: {total_rows}")
```

```
Number of rows with PF values of 6 or more: 446
Percentage of rows with PF values of 6 or more: 0.42%
Total number of rows: 105622
```

4. Methods

We established a baseline comparison model that used only the current data, which inadvertently resulted in a data leakage issue. However, this model was primarily utilized for comparison purposes with other models, allowing us to better understand how the attributes contributed to the predictions and highlighting the importance of incorporating historical context for more robust analysis.

Two types of predictive models were employed in this analysis: regression and classification models. Regression models were used to predict the number of personal fouls (PF) as a continuous variable. Among these, the Decision Tree Regression model provided interpretable rules for understanding fouling behavior. On the other hand, classification models aimed to predict whether a player would foul out (Fouled_Out = 1) or not (Fouled_Out = 0). Various models, including Decision Tree and Ensemble models like Random Forest and Gradient Boosting, were tested to improve performance.

```python
# Define predictors and target
X_reg = merged_w_3[['MIN_avg_3', 'FGA_avg_3', 'FG_PCT_avg_3', 'FG3A_avg_3',
                    'FG3_PCT_avg_3', 'FTA_avg_3', 'FT_PCT_avg_3', 'OREB_avg_3', 'DREB_avg_3',
                    'AST_avg_3', 'STL_avg_3', 'BLK_avg_3', 'TO_avg_3', 'PLUS_MINUS_avg_3']]
y_reg = merged_w_3['PF']  # Target: Predicting Personal Fouls
```

[Regression Model]

```python
# Create binary target for fouling out
merged_w_noNA['Fouled_Out'] = (merged_w_noNA['PF'] >= 6).astype(int)

# Define predictors and target
X_cl = merged_w_noNA[['MIN_avg_3', 'OREB_avg_3', 'PTS_avg_3', 'REB_avg_3',
                      'AST_avg_3', 'STL_avg_3', 'BLK_avg_3', 'TO_avg_3']]
y_cl = merged_w_noNA['Fouled_Out']
```

[Classification Model]

The performance of these models was evaluated using metrics such as accuracy, precision, recall, and F1-score.

Ensemble models, particularly Random Forest and Gradient Boosting, were designed to enhance prediction performance by addressing class imbalances. Random Forest trains multiple decision trees in parallel and averages their predictions to create a more robust and generalized model. In contrast, Gradient Boosting trains trees sequentially, with each tree learning from the mistakes of the previous ones to refine the overall prediction. These methods leverage the strengths of multiple trees to improve effectiveness. Techniques such as SMOTE oversampling were also employed to create a more balanced dataset, helping the models handle skewed data more effectively. Additionally, hyperparameter tuning was applied to optimize the models, although it did not lead to significant improvements in performance for this dataset. Together, these approaches provided valuable insights into fouling out behavior and the challenges of modeling imbalanced data.

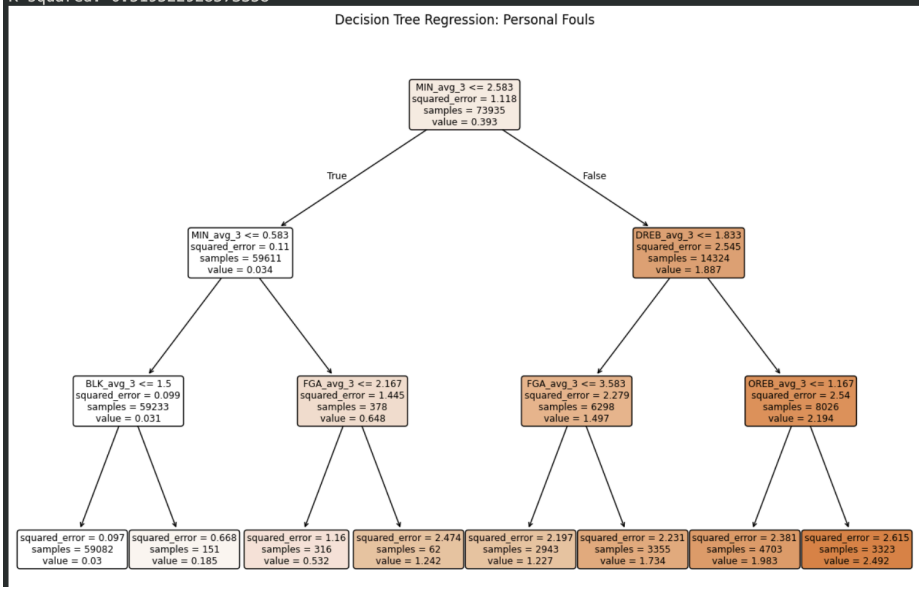
5. Results

Note: the most accurate models are the models that run on “merged_w_3” dataframe, the ones that run on “merged_w_noNA” should be ignored as they include NA rows

The models were then run using the following variables: 'MIN_avg_3', 'OREB_avg_3', 'PTS_avg_3', 'REB_avg_3', 'AST_avg_3', 'STL_avg_3', 'BLK_avg_3', 'TO_avg_3'. The Y target variable was either set to Personal Fouls ‘PF’ for the Regression Models and a binary ‘Fouled_Out’ variable for the Classification models, created as a new column with the value as 1 for all players with PF >= 6, otherwise as a 0.

Our initial Regression Decision Tree did not perform well, with an R-squared value of 0.5 and an RMSE of 0.73. This indicates that, on average, predictions were nearly 0.75 fouls off the actual count, making the model not particularly useful. It also failed to predict players committing more than three fouls, despite a notable minority exceeding this threshold, especially those who fouled out. The primary splits were based on minutes played, followed by defensive rebounds and field goal average. Min_avg_3 had a 98% importance ranking, making it by far the most significant variable, while most others contributed negligibly, with near-zero importance scores.

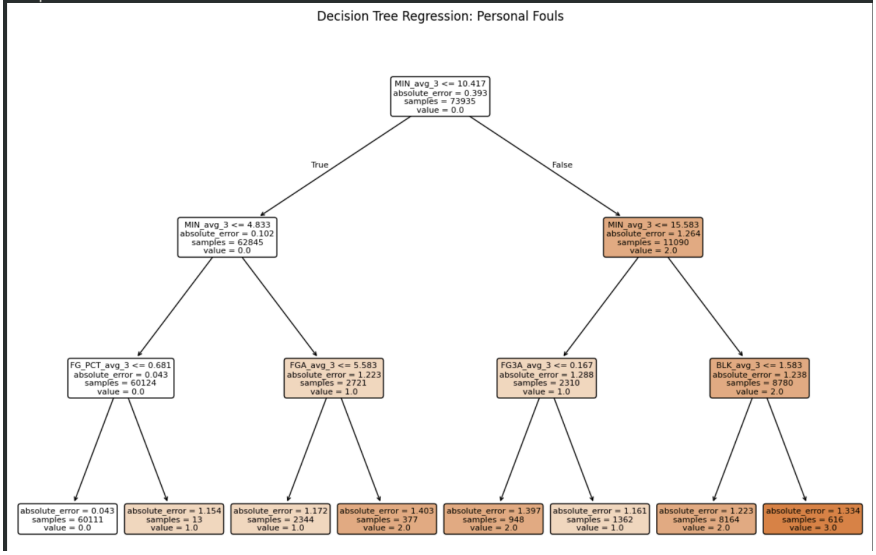
```
Mean Squared Error (MSE): 0.5357262566577485
Root Mean Squared Error (RMSE): 0.7319332323769352
R-squared: 0.519522928573358
```




```
Feature Importances:
             Feature  Importance
0          MIN_avg_3    0.980019
3         FG3A_avg_3    0.007361
11         BLK_avg_3    0.006730
1          FGA_avg_3    0.005574
2       FG_PCT_avg_3    0.000315
4      FG3_PCT_avg_3    0.000000
5          FTA_avg_3    0.000000
6       FT_PCT_avg_3    0.000000
7         OREB_avg_3    0.000000
8         DREB_avg_3    0.000000
9          AST_avg_3    0.000000
10         STL_avg_3    0.000000
12          TO_avg_3    0.000000
13  PLUS_MINUS_avg_3    0.000000
```

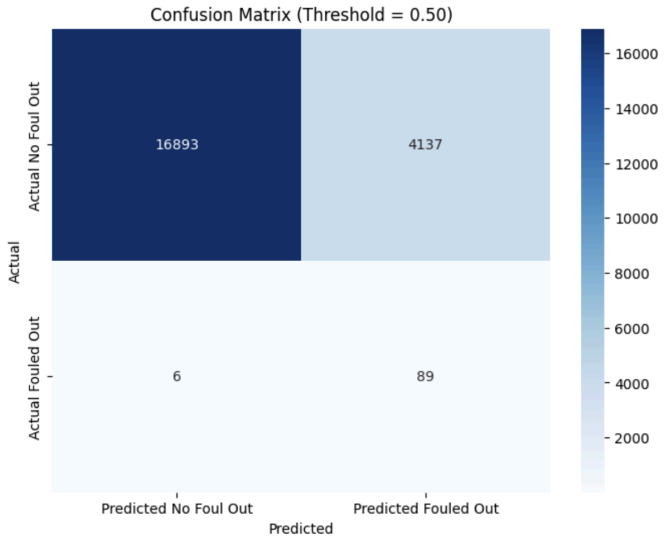
In an adjusted variant adjusting the minimum samples in a leaf and the minimum samples needed to split actually made R squared decrease to 0.49. Some of the lower variable splits changed (such as the first split from FALSE from DREB to MIN) but importance rankings were almost identical.

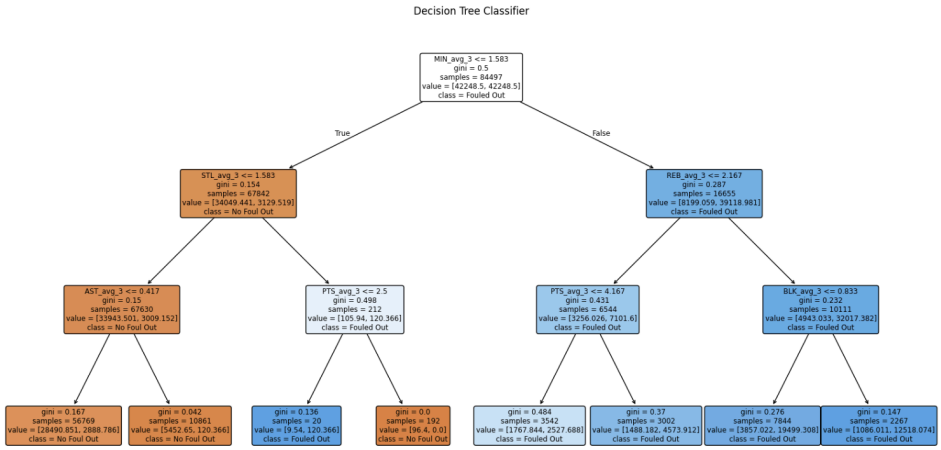
```
Mean Squared Error (MSE): 0.5602297472149462
Root Mean Squared Error (RMSE): 0.7484849679285124
R-squared: 0.4975464709397457
```



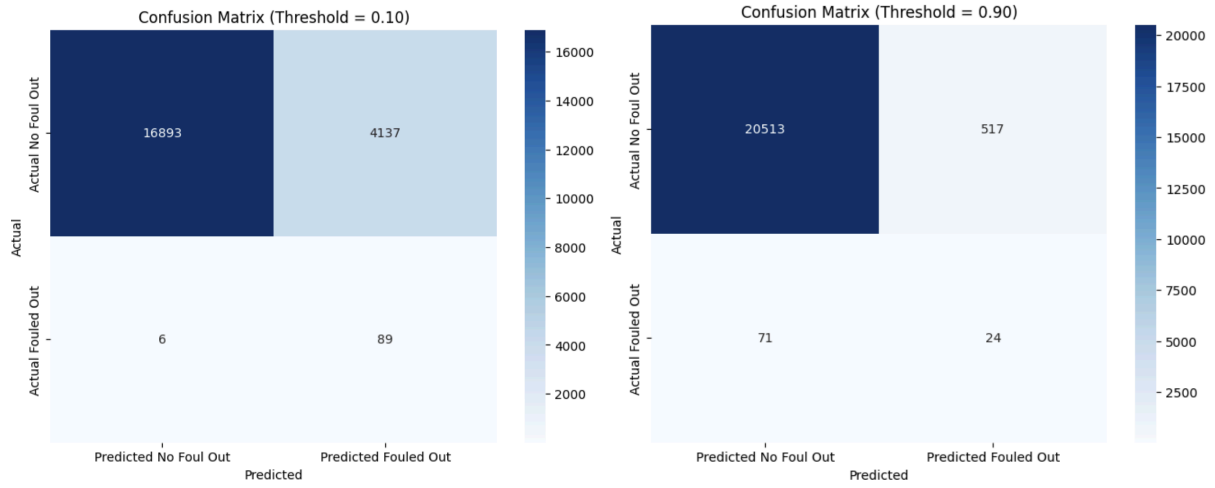
We then moved on to classification models, but overall performance was poor across all models tested, with minimal improvements despite adjusted hyperparameters in variant models. Performance was evaluated using the F1 score and its components, precision and recall. To address class imbalance, we set class_weight='balanced' for all models, though additional handling was still required.

In the baseline single decision tree with a threshold of 0.5, the model performed poorly, achieving an F1 score of just 0.04. Precision was near 0, while recall was close to 1. This indicates that the model successfully identified nearly all true positives (players who fouled out) but misclassified many non-foul-out players as fouling out, resulting in a high number of false positives. This issue is evident in the top right corner of the classification model's confusion matrix.

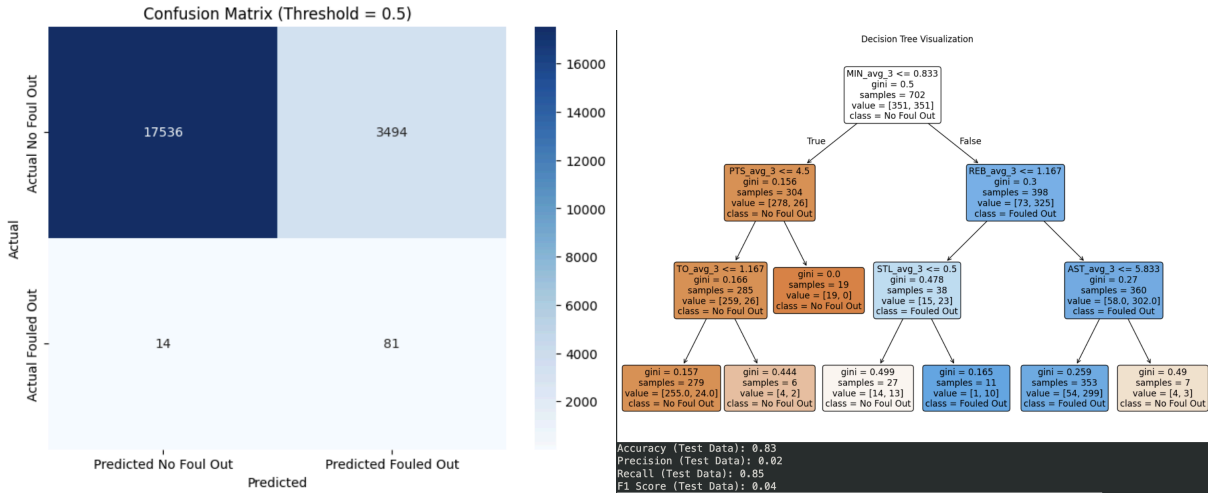




We also tested different decision thresholds but found no significant improvement in the single-tree model. Adjusting the threshold alters the criteria the tree uses to classify a data point as “positive.” Lowering the threshold led to a substantial increase in false positives, where players who did not foul out were incorrectly labeled as fouling out, without improving true positive classification. Conversely, raising the threshold caused the opposite issue, broadly classifying more players as not fouling out, even excluding some true positives that were previously identified.



Attempting to improve the model by adding random undersampling and hyperparameter with the default value led to no significant improvement, as the F1 score stayed the same.

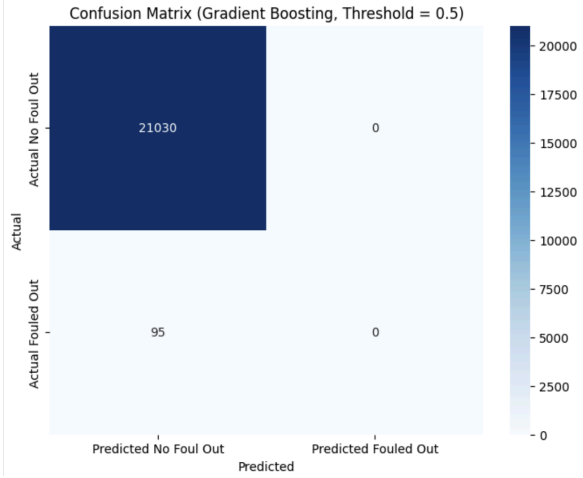


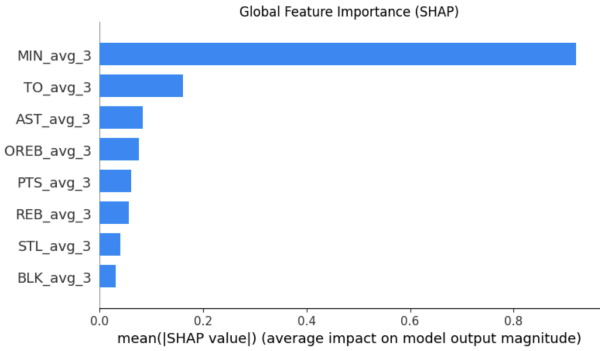

```python
# Apply Random Undersampling to the training data
undersampler = RandomUnderSampler(random_state=42)
X_train_undersampled, y_train_undersampled = undersampler.fit_resample(X_train_cl, y_train_cl)

# Train decision tree classifier with hyperparameter adjustments
tree_cl_undersampled = DecisionTreeClassifier(
    random_state=42,
    max_depth=3,              # Limit tree depth to control overfitting
    min_samples_split=10,     # Require at least 10 samples to split a node
    min_samples_leaf=5,       # Require at least 5 samples in a leaf
    ccp_alpha=0.000,           # Cost-complexity pruning to simplify the tree
    class_weight=None         # Not needed due to undersampling
)
tree_cl_undersampled.fit(X_train_undersampled, y_train_undersampled)
```

We then attempted to build ensemble models to see if the use of multiple trees could improve the quality of our predictions. We first tried to use a base Gradient Boosting model. However, the first gradient boosting model completely pivoted and predicted all players as not fouling out. This lead to a very high accuracy score, as it reflected how over 99% of the data were not foul-outs, but lead to a 0 score for Precision, Recall, and F1 score.

For the ensemble models, since they are made with a combination of multiple trees there is no single tree display that can be shown, but we can use feature importance graphs to attempt to determine which features were most important (and therefore most likely and commonly used in the trees). The clear most important feature was MIn_avg_3

```
merged_w_3['Fouled_Out'] = (merged_w_3['PF'] >= 6).astype(int)
Gradient Boosting - Accuracy: 1.00
Gradient Boosting - Precision: 0.00
Gradient Boosting - Recall: 0.00
Gradient Boosting - F1 Score: 0.00
```

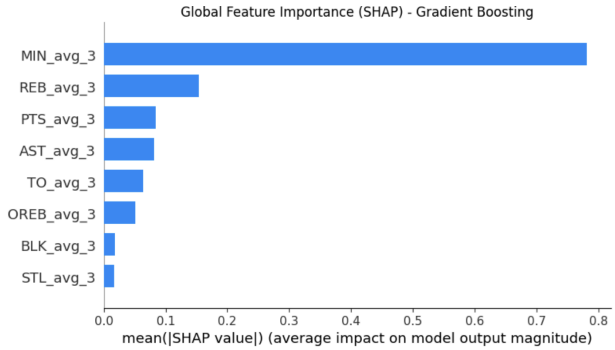





```
pproximate Decision Rules for Sample:
f MIN_avg_3 = 0.00, it contributes -0.59 to the prediction.
f TO_avg_3 = 0.00, it contributes -0.13 to the prediction.
f AST_avg_3 = 0.00, it contributes 0.06 to the prediction.
f OREB_avg_3 = 0.00, it contributes -0.05 to the prediction.
f PTS_avg_3 = 0.00, it contributes 0.03 to the prediction.
f REB_avg_3 = 0.00, it contributes -0.02 to the prediction.
f BLK_avg_3 = 0.00, it contributes -0.02 to the prediction.
f STL_avg_3 = 0.00, it contributes -0.00 to the prediction.
```

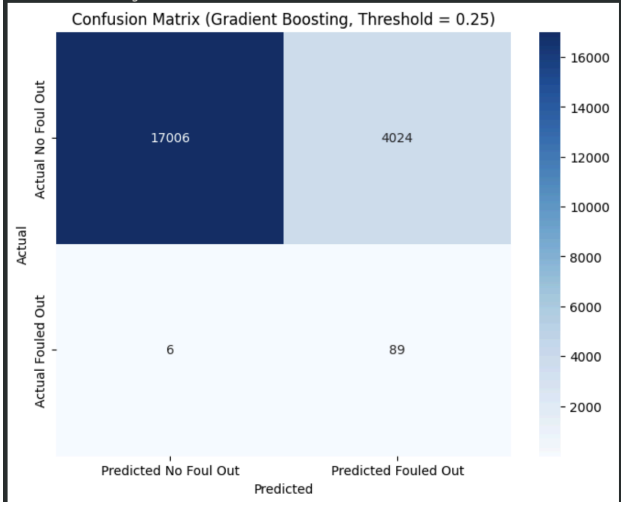
```python
# Train Gradient Boosting Classifier
gb_clf = GradientBoostingClassifier(
    random_state=42,
    n_estimators=100,       # Number of boosting stages
    learning_rate=0.1,      # Shrinks contribution of each tree
    max_depth=3,            # Maximum tree depth
    min_samples_split=10,   # Minimum samples to split a node
    min_samples_leaf=5      # Minimum samples in a leaf
)
gb_clf.fit(X_train_gb, y_train_gb)

# Predict probabilities on test data
y_pred_probs_gb = gb_clf.predict_proba(X_test_gb)[:, 1]

# Adjust the classification threshold
threshold = 0.5
y_pred_gb = (y_pred_probs_gb >= threshold).astype(int)
```

Attempting to fix this in the variant model, setting class_weight to balanced and adding more trees and smaller splits lead to a slight F1 score increase to 0.04, to about the same level as the basic single decision tree.




```
pproximate Decision Rules for Sample:
f MIN_avg_3 = 0.00, it contributes -0.49 to the prediction.
f REB_avg_3 = 0.00, it contributes -0.09 to the prediction.
f PTS_avg_3 = 0.00, it contributes 0.08 to the prediction.
f AST_avg_3 = 0.00, it contributes 0.04 to the prediction.
f OREB_avg_3 = 0.00, it contributes -0.03 to the prediction.
f TO_avg_3 = 0.00, it contributes -0.02 to the prediction.
f BLK_avg_3 = 0.00, it contributes 0.01 to the prediction.
f STL_avg_3 = 0.00, it contributes -0.00 to the prediction.
```

```
Gradient Boosting - Accuracy: 0.81
Gradient Boosting - Precision: 0.02
Gradient Boosting - Recall: 0.94
Gradient Boosting - F1 Score: 0.04
```

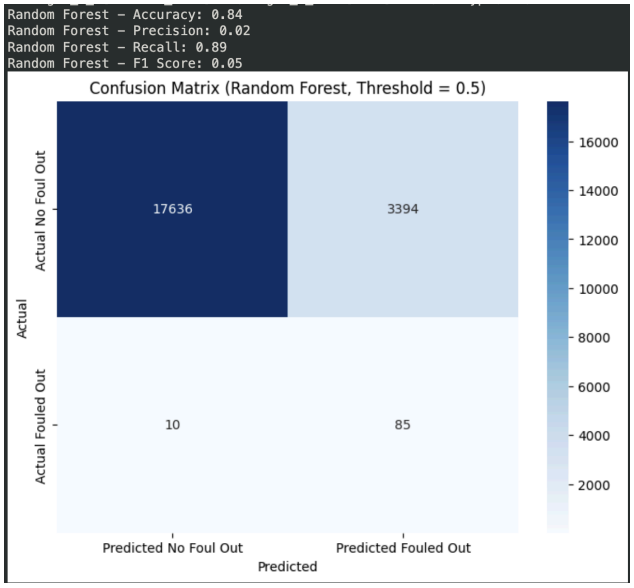



```python
#adjustments to hyperparmeters and lowering threshold classification to favorm inority

# Calculate class weights
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train_gb),
    y=y_train_gb
)
sample_weights = np.where(y_train_gb == 0, class_weights[0], class_weights[1])

# Train Gradient Boosting Classifier
gb_clf = GradientBoostingClassifier(
    random_state=42,
    n_estimators=200,       # More trees for better learning
    learning_rate=0.01,     # Slower learning rate
    max_depth=5,            # Moderate depth
    min_samples_split=5,    # Smaller splits
    min_samples_leaf=2      # Smaller leaves
)
gb_clf.fit(X_train_gb, y_train_gb, sample_weight=sample_weights)

# Predict probabilities on test data
y_pred_probs_gb = gb_clf.predict_proba(X_test_gb)[:, 1]

# Adjust the classification threshold
threshold = 0.25  # Lower threshold to favor the minority class
y_pred_gb = (y_pred_probs_gb >= threshold).astype(int)
```

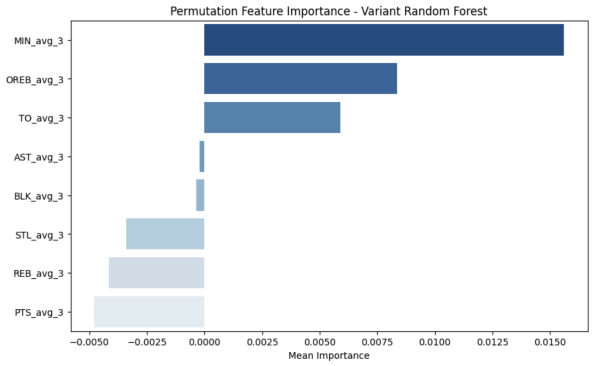
We then moved on to the Random Forest ensemble model. Running the base model lead to heavy false positive misclassification but an F1 score of 0.05, about as strong as the best basic tree.

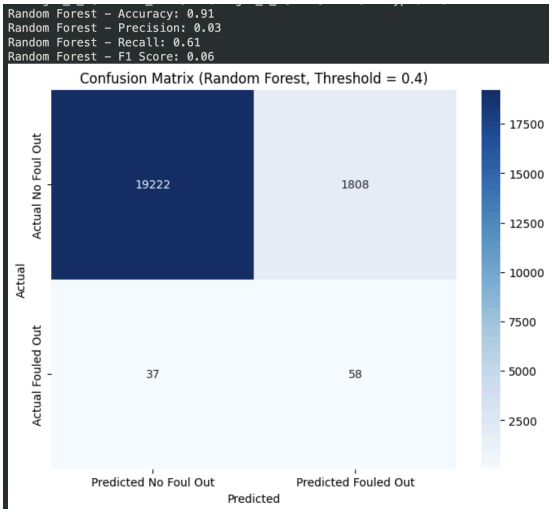
```python
# Train Random Forest Classifier
rf_clf = RandomForestClassifier(
    random_state=42,
    n_estimators=100,       # Number of trees
    max_depth=5,            # Maximum tree depth
    min_samples_split=10,   # Minimum samples to split a node
    min_samples_leaf=5,     # Minimum samples in a leaf
    class_weight='balanced' # Balances class weights
)
rf_clf.fit(X_train_rf, y_train_rf)

# Predict probabilities on test data
y_pred_probs_rf = rf_clf.predict_proba(X_test_rf)[:, 1]

# Adjust the classification threshold
threshold = 0.5
y_pred_rf = (y_pred_probs_rf >= threshold).astype(int)
```

```
Random Forest - Accuracy: 0.84
Random Forest - Precision: 0.02
Random Forest - Recall: 0.89
Random Forest - F1 Score: 0.05
```



In the variant model, we attempted to adjust the hyperparameters (including slightly lowering the decision threshold to 0.4), and include SMOTE minority class (foul-out class) oversampling to adjust for class imbalance, This resulted in the best F1 score of all of the models of 0.06 but it was still poor, Minutes again is still most important factor according to SHAP.



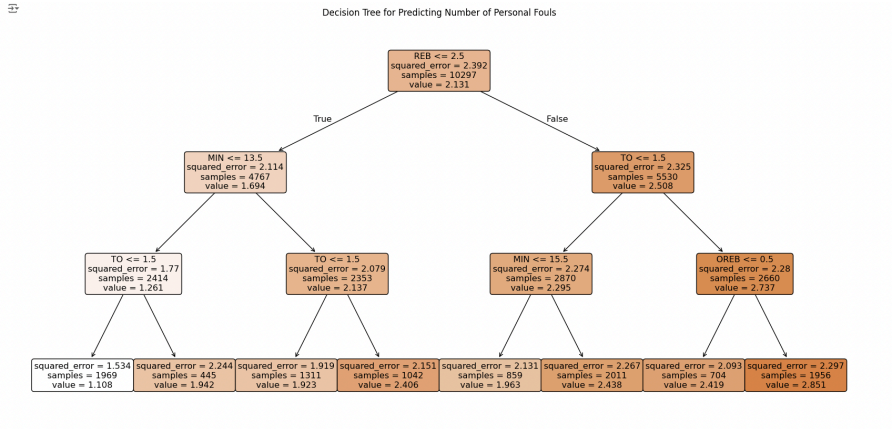



```python
# Apply SMOTE to balance the dataset
smote = SMOTE(random_state=42)
X_train_rf_smote, y_train_rf_smote = smote.fit_resample(X_train_rf, y_train_rf)

# Train Random Forest Classifier with tuned parameters
rf_clf = RandomForestClassifier(
    random_state=42,
    n_estimators=300,         # Number of trees in the forest
    max_depth=10,             # Maximum tree depth
    min_samples_split=5,      # Minimum number of samples to split a node
    min_samples_leaf=2,       # Minimum number of samples in a leaf
    class_weight='balanced_subsample'  # Balance classes at tree level
)
rf_clf.fit(X_train_rf_smote, y_train_rf_smote)

# Predict probabilities on test data
y_pred_probs_rf = rf_clf.predict_proba(X_test_rf)[:, 1]

# Adjust the classification threshold
threshold = 0.4
y_pred_rf = (y_pred_probs_rf >= threshold).astype(int)
```

Because all of our models performed poorly we decided to compare our models that used previous game data to predict with a “current” game model. This meant using the regular variables (OREB vs OREB rolling average over the past 3 games, etc.) to see if the current stats from the same game even predicted the current game’s PF well. Both baseline “leakage” models where we used the current game’s box scores to predict the current game’s fouls also performed poorly and worse than our base decision tree models (R squared 0.11 and F1 score

0), suggesting that even in a case where the variables directly contributed to PF 1:1 they do not predict foul behaviors strongly.

```
# Step 2: Select Predictors for Decision Tree
# Filter valid data and select predictors for initial modeling. Include all potential predictors for decision tree analysis.

# Filter rows with valid data
filtered_data = merged_w_noNA[(merged_w_noNA['MIN'] > 0)]

# Select predictors and target
X_all = filtered_data[['MIN', 'OREB', 'PTS', 'REB', 'AST', 'STL', 'BLK', 'TO']]  # Example predictors
y = filtered_data['PF']
```




```
# prompt: calculate R squared for the decision tree model above

from sklearn.metrics import r2_score

# Assuming 'dt_model' and 'X_test_dt', 'y_test_dt' are defined from the previous code
y_pred_dt = dt_model.predict(X_test_dt)
r2 = r2_score(y_test_dt, y_pred_dt)
print(f"R-squared for the Decision Tree model: {r2}")
```

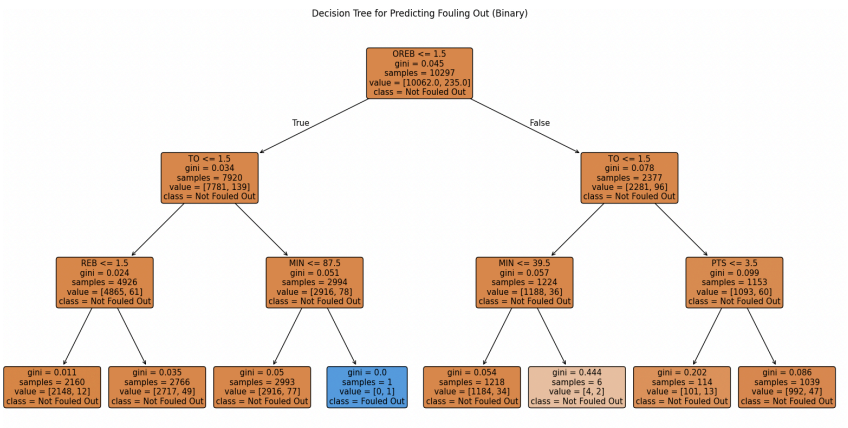
```
R-squared for the Decision Tree model: 0.11926520699059906
```

```
# Extract feature importance
importances = pd.DataFrame({
    'Feature': X_all.columns,
    'Importance': dt_model.feature_importances_
}).sort_values(by='Importance', ascending=False)

print(importances)
```

```
  Feature  Importance
3     REB    0.484996
0     MIN    0.299715
7      TO    0.187660
1    OREB    0.027629
2     PTS    0.000000
4     AST    0.000000
5     STL    0.000000
6     BLK    0.000000
```

Classification

```
# Step 3: Select Predictors for Decision Tree
# Filter rows with valid data
filtered_data = merged_w_noNA[(merged_w_noNA['MIN'] > 0)]

# Create a binary target variable for fouling out (1 = Fouled Out, 0 = Not Fouled Out)
filtered_data['Fouled_Out'] = (filtered_data['PF'] >= 6).astype(int)

# Select predictors and the new binary target
X_all = filtered_data[['MIN', 'OREB', 'PTS', 'REB', 'AST', 'STL', 'BLK', 'TO']]  # Predictors
y = filtered_data['Fouled_Out']  # Binary target variable
```




```
Accuracy: 0.98
Precision: 0.00
Recall: 0.00
F1-Score: 0.00
Confusion Matrix:
[[1509    1]
 [  31    0]]
```

Key Findings:

We evaluated player-game box score data across various models to predict players' foul counts in future games but found that the models performed poorly, even after testing variants to improve their performance. Regression trees demonstrated weak correlation values of 0.5 or less, with RMSE values showing predictions were off by nearly 0.75–1 foul on average, making them unsuitable for real-life applications. Classification models also underperformed, with poor precision and F1 scores. The highest F1 score was 0.06 from the Random Forest variant and 0.05 from the adjusted Gradient Boosting and single-tree models. Given the scale of F1 scores (0–1), this level of performance, combined with frequent misclassification of players as false positives or false negatives, renders these models ineffective for practical use.

A consistent trend across all models was the dominance of minutes played (or average minutes across the past three games) as the most important factor, often serving as the first splitting criterion. This aligns with intuition since players must be active in the game to commit fouls. Most other variables had near-zero importance scores and were rarely used for splits, explaining why increasing tree size in single and ensemble models failed to improve performance. Additionally, the significant class imbalance (with few players fouling out or committing more than four fouls) severely impacted performance. Despite efforts to address this through techniques like class weighting, hyperparameter tuning, and over/undersampling, none significantly improved results.

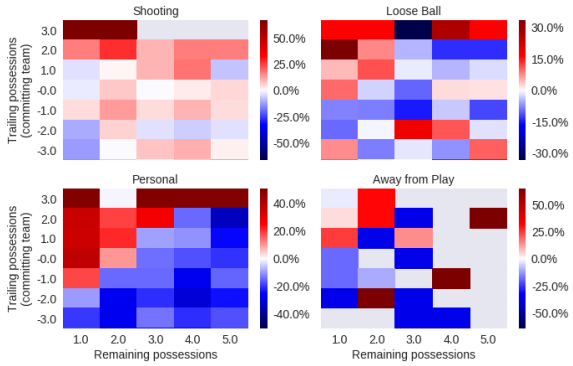
From these findings, we conclude that a strong single-tree regression model would be ideal due to its interpretability and ability to provide more actionable foul count predictions. However, even

this approach is limited by the inadequacy of the dataset. The current player-game box score data is insufficient for predicting fouls, and better data would be required to build effective models. Future exploration could involve using past foul counts as a predictor variable or adjusting the range of rolling averages (e.g., past 10 games). Nevertheless, the poor performance of current-game “leakage” models compared to baseline past-data models reinforces the conclusion that this data, whether past or present, is not predictive enough for foul behavior.

6. Conclusion

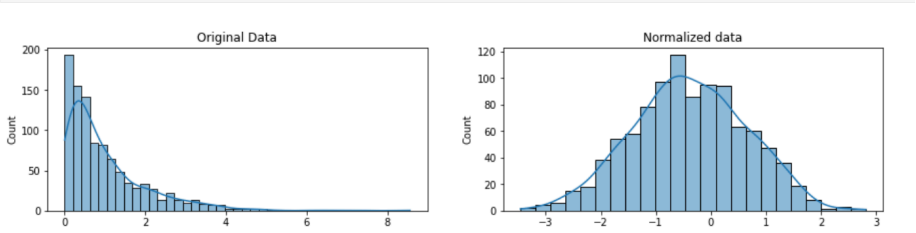
For the primary stakeholder, our results and findings indicate that this model requires further refinement and is not recommended for use at this time. The analysis performed above shows that NBA box score data is not an effective predictor when it comes to predicting personal fouls in future games. Although a direct answer to the problem statement was achieved, future steps and recommendations can be proposed.

The current model relies solely on box score data which evidently proved insufficient for prediction of personal fouls. It would be worth reworking the study with some data enhancements such as including player characteristic data like height and wingspan that may have a higher correlation with personal fouls. Along with this, studying substitution data and spatial data in the form of foul maps would almost certainly aid in predicting fouls for a player. Other than data enhancements, looking into new data sources may also improve the study as box score data has many limitations. Other data sources include “Close game data” that include specific foul types like shooting fouls, loose ball fouls, away from play fouls, and personal fouls.



In addition to using new sources and data, one recommendation to help optimize the study would be the use of feature engineering. Feature engineering can be defined as the transformation of certain raw variables/inputs into more relevant features in order to improve model performance. Combining and adjusting data through feature engineering has the potential to provide more accurate data by reducing complexity, noise, and excess redundant information. Some examples include standardizing variables, scaling them differently, using

functions of certain values (log), Principle Component Analysis (PCA), and even variable combinations.



Based on all the results and analysis above, it may be a good idea to reevaluate and rephrase the initial problem statement. Instead of predicting the number of fouls committed by a player, it may be a better idea to predict the **impact** of a foul, whether that be valuable or detrimental. The current model lacks context of fouls and treats all fouls as if they are equal. Fouls can be offensive, defensive, tactical, unintentional and more, so differentiating between these types of fouls and their implications may help increase the accuracy of our models.

Finding more specific data and asking a more specific research question like: “How does the number of offensive fouls committed by a player affect play-by-play player performance?” will lead to more effective research and results in the future.