(PDF version of the methodology page in our webpage)

Methodology

To enable backtesting across multiple seasons, we restricted our training data to the 2023–24 NBA season. Due to this constraint and the relatively limited volume of game data available via the API – insufficient for effectively training a neural network at least – we chose not to use a time-series-based deep learning approach. Instead, we implemented a structured, fold-based machine learning pipeline centered around a stacked ensemble model for binary classification. This ensemble method leverages the strengths of multiple base models by passing the output of one as input to the next, allowing each model to correct errors made by its predecessor.

Out of the original 20 basketball statistics, five were redundant because they directly contributed to the calculation of others—either as numerators, denominators in percentage metrics (e.g., FGA and FGM are included in FG_PCT), or as components summed to form aggregate stats. These five redundant features were therefore removed to streamline the dataset. Further, To better structure our features for ensemble learning, we engineered four time-aware statistics – AVG, MAX, BIAS, and MOM – for each of the 15 core basketball statistics (explained in detail on the glossary page). These were computed per game instance in the dataset as follows:

- AVG: The average value of the stat prior to the game date, excluding the game itself (to preserve prediction integrity).
- MAX: The maximum value observed for that stat before the game date.
- BIAS and MOM: Derived from a simple least squares linear regression fit to the stat's historical values – BIAS represents the intercept (a baseline), and MOM (momentum) represents the slope (the trend). These were included to capture directional trends over time.

After generating these features, the remaining original 15 raw stats were dropped, leaving us with 60 engineered features per team per game instance. Since basketball game outcomes are inherently relative rather than absolute, we appended the opposing team's 60 features (feature names appended with _OPP indicate values corresponding to the opposing team) to the same row, always listing the home team's features first. This transformation halved the number of examples but significantly improved their informational quality, resulting in 120 features per instance. The data instances were chronologically ordered, with the earliest game appearing first in the dataset. This ensured temporal integrity and prevented data leakage from future information influencing past predictions.

This constituted our input feature dataset X. The corresponding label vector y was a binary indicator: 1 if the home team won the game, 0 otherwise. The original dataset used to construct the input matrix X and target vector y is available for download via the Glossary page.

The overall workflow consisted of:

- Data preprocessing (described above),
- Feature engineering and selection (guided by correlation analysis),
- Model training via ensemble methods with threshold optimization to calibrate predicted probabilities,
- Testing and analysis of results.

A detailed breakdown of the fold-level training and evaluation process follows below.

1. Data Partitioning

For each fold, the dataset was split into training and testing sets using predefined indices (train_index and test_index). For each successive fold, the size of the training set increased incrementally – starting with the earliest game instances and progressing chronologically. This setup simulated training on distinct datasets per fold and helped improve the ensemble model's robustness against overfitting by exposing it to varying temporal patterns across folds. The training set was further divided into a training and validation set using an 80/20 split, with shuffle=False to preserve any temporal structure present in the data. This partitioning ensured that model validation occurred on unseen data within each fold.

2. Missing Value Imputation

We computed the mean of each feature in the training set and imputed missing values in the training, validation, and test sets using these means. This approach avoided data leakage by preventing the validation or test sets from influencing the imputation process.

3. Feature Scaling

All features were standardized using StandardScaler, which centers and scales features to have zero mean and unit variance. The scaler was fitted exclusively on the training data and then applied to the validation and test sets to ensure consistency and avoid leakage.

4. Feature Selection via Correlation Analysis

To reduce multicollinearity and improve model generalization, we calculated the pairwise Pearson correlation matrix of the scaled training features. Features with absolute correlation values exceeding a threshold magnitude of 0.7 were dropped. This was done prior to model training and consistently applied to the validation and test sets.

5. Model Architecture: Stacked Ensemble

We implemented a stacked ensemble classifier using the following base learners:

- Random Forest Classifier: robust, non-parametric models that perform well on structured tabular data. They reduce overfitting through bootstrap aggregation (bagging) and capture complex feature interactions without extensive preprocessing. We included it for its strong baseline performance and its ability to handle noisy, high-dimensional data effectively.

- XGBoost Classifier: a high-performance gradient boosting algorithm known for its accuracy and efficiency. It handles missing data internally and is well-suited for tabular, imbalanced, or sparse datasets. Using log-loss as the evaluation metric allows it to optimize directly for probabilistic outputs, which is beneficial when calibrating thresholds or combining models in an ensemble.

- Support Vector Classifier (SVC): effective in high-dimensional spaces and can model non-linear decision boundaries through kernel tricks. By enabling probability estimates, the SVC contributes calibrated predictions that complement the outputs of tree-based models. Its inclusion enhances the ensemble's diversity, particularly when separating data that is not linearly separable.

- k-Nearest Neighbors (kNN) Classifier: kNN is a simple yet powerful instance-based learner that makes predictions based on local proximity in feature space. While sensitive to feature scaling and noisy data, its fundamentally different approach – non-parametric and lazy – adds a contrasting inductive bias to the ensemble. It can capture local structure that other global learners may miss.

A logistic regression model served as the final meta-learner. Its role was to learn optimal weights for combining the base learners' probabilistic outputs into a final prediction. The stacking classifier used passthrough=True to retain original features alongside base learner outputs and was trained using 2-fold cross-validation within the training set to optimize internal base model performance.

6. Threshold Tuning

To convert predicted probabilities into binary class labels, we calibrated the decision threshold (predictions were assigned a value of 1 if the predicted probability met or exceeded the threshold; otherwise, the prediction was 0) using the validation set. We evaluated thresholds in the range [0.30, 0.69] with a step size of 0.01 and selected the classification threshold that maximized the balanced accuracy score, which accounts for potential class imbalance. This approach allowed us to favor higher recall – crucial in the context of sports betting, where minimizing false negatives (i.e., missing potential wins) is often more valuable than maximizing precision.

7. Model Persistence

All relevant fold-specific artifacts, including feature metadata, preprocessing parameters (mean values, scaler), dropped features, the trained stacking model, and the optimal classification threshold, were saved using joblib for reproducibility and downstream inference.

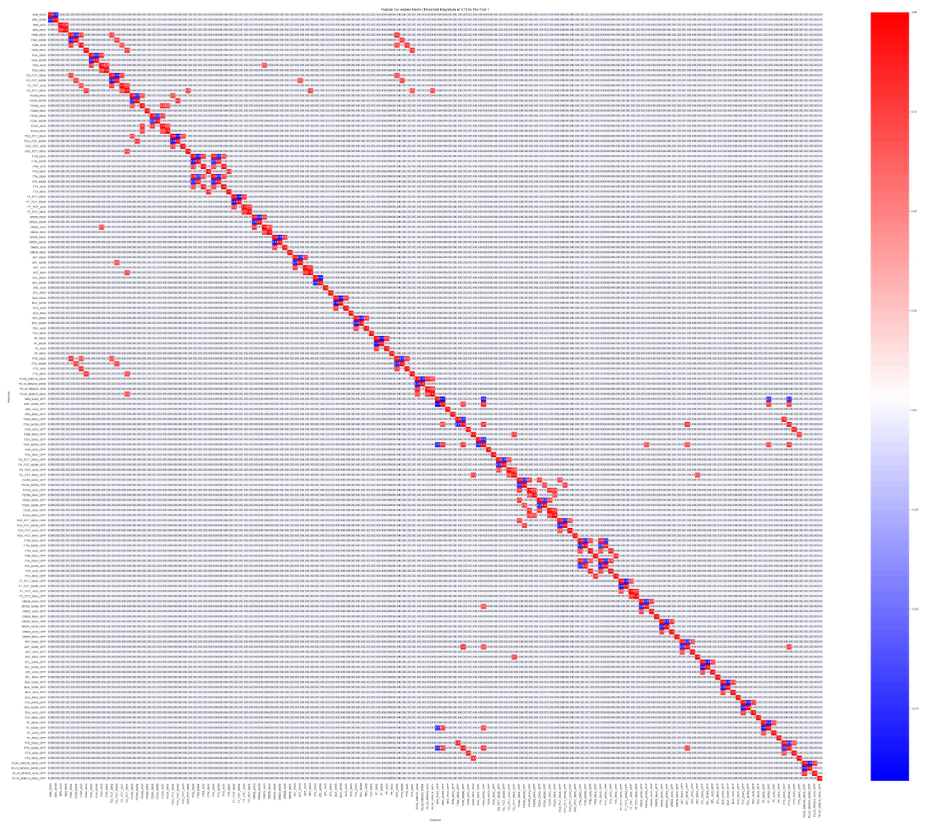
8. Model Predictions

To generate predictions, each data instance was passed through all 10 folds of the ensemble pipeline, producing one prediction per fold. The final prediction for a given game was determined by taking the mode of these 10 outputs. In the event of a tie (e.g., five predictions of 0 and five of 1), the final prediction defaulted to 0, as per the behavior of scipy.stats.mode, which returns the smallest value in the case of a tie.

Results & Discussion

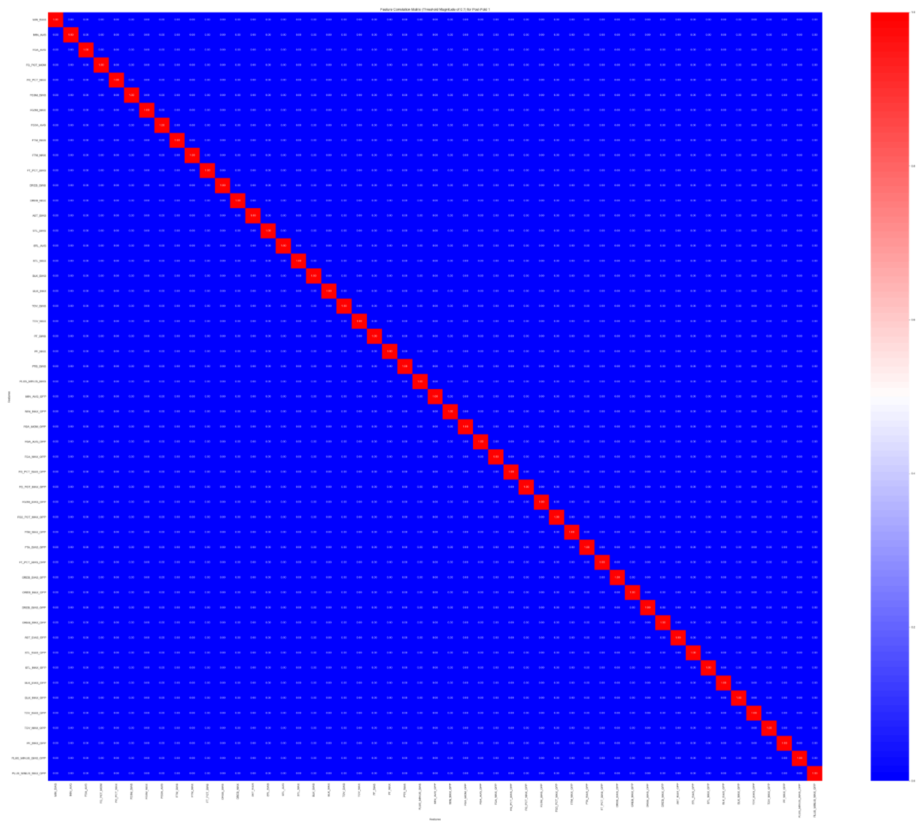
Here, we analyze and break down the results from a single fold – specifically, fold 1 – and present the averages across all folds for each statistic. Furthermore, we include backtesting results for the 2024–25 NBA season. Additional images for your review and analysis are provided in the appendix below.

Below is the correlation matrix for the features in fold 1's dataset before feature selection. With 120 features, the matrix is dense and can be challenging to interpret in detail. The key insight lies in the overall density and distribution of strong correlations, which is visually apparent. To enhance clarity, all correlation values below the 0.7 threshold were set to zero and displayed as white cells, simplifying the visualization and highlighting only the most significant feature relationships.



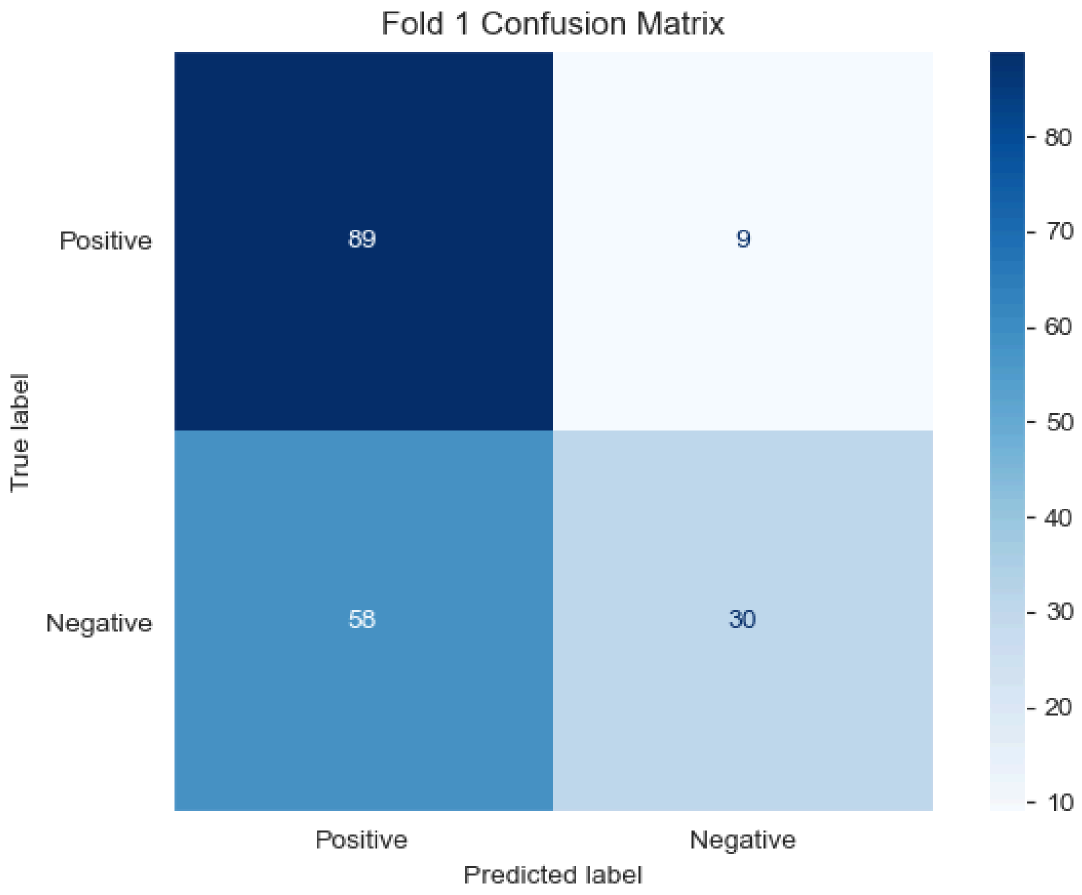
It is evident that most correlations cluster near the diagonal, indicating strong relationships among closely related features – such as DREB_BIAS and DREB_AVG – which is expected. However, this approach also uncovered less obvious correlations (the ones not near the diagonal), such as between PTS_MAX_OPP and FGM_MAX_OPP, highlighting interactions that might otherwise have been overlooked.

After removing all redundant features, 53 remain. This reduction is reflected visually in the much sparser correlation matrix below. Notice that aside from the trivial diagonal (highlighted in red), which represents the perfect self-correlation of each feature, no correlations exceed the 0.7 threshold (indicated by blue). This confirms the effective elimination of highly correlated, redundant features.



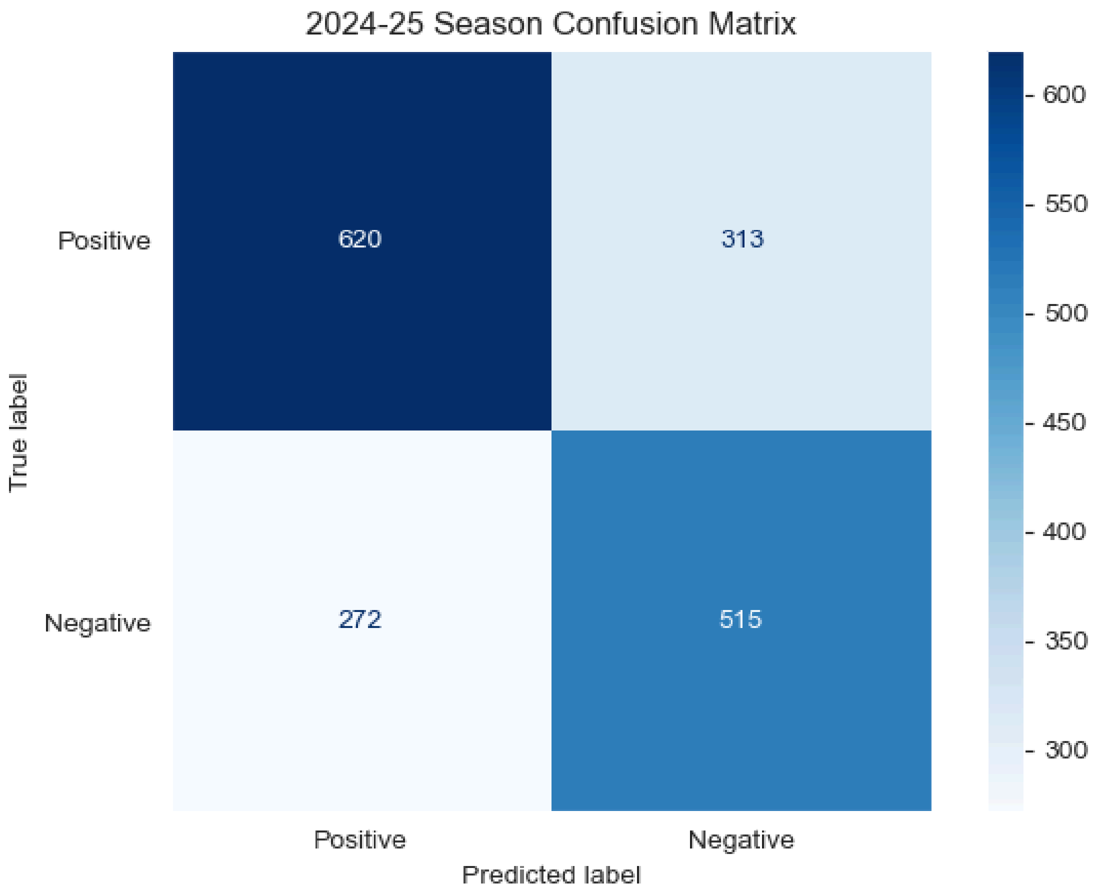
By retaining only about 40% of the original features, we significantly reduce training and prediction times while maintaining most of the model’s predictive power.

Additionally, the confusion matrix for fold 1 is presented below:



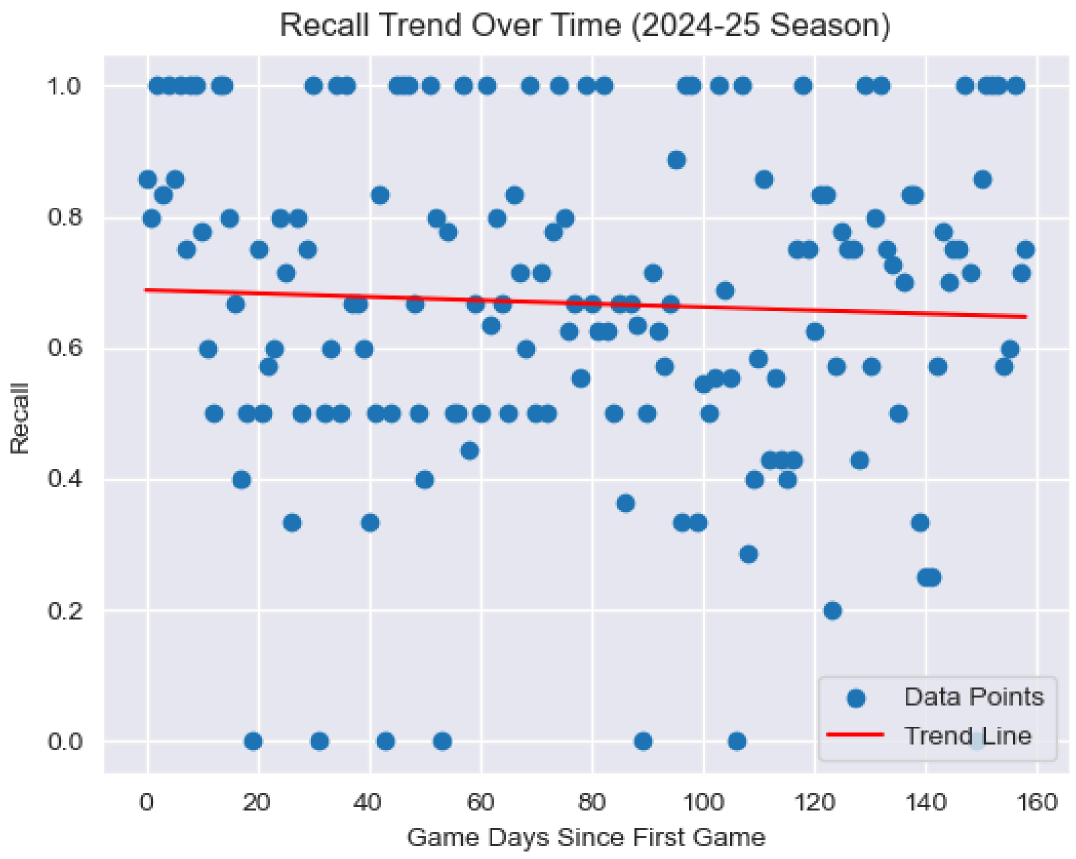
As shown, there are 89 true positives – cases where the model correctly predicted a win (1) and the game outcome was indeed a win – along with 30 true negatives, 58 false positives, and 9 false negatives. From this confusion matrix, fold 1's recall is 0.91, precision 0.61, F1 score 0.73, and accuracy 63.98%. While these results appear promising, they should be interpreted with caution since fold 1 was evaluated on a relatively small test set, making the metrics susceptible to volatility. This limitation motivated backtesting on a larger dataset, such as all games during the previous NBA season, to gain more knowingly stable estimates. Despite this, fold 1's performance remains a useful indicator of training behavior and results. Across all 10 folds, the average metrics were: recall at 0.72, precision 0.64, F1 score 0.67, and accuracy 62.53%, with an average optimal classification threshold of 0.5670 – above the standard threshold of 0.5.

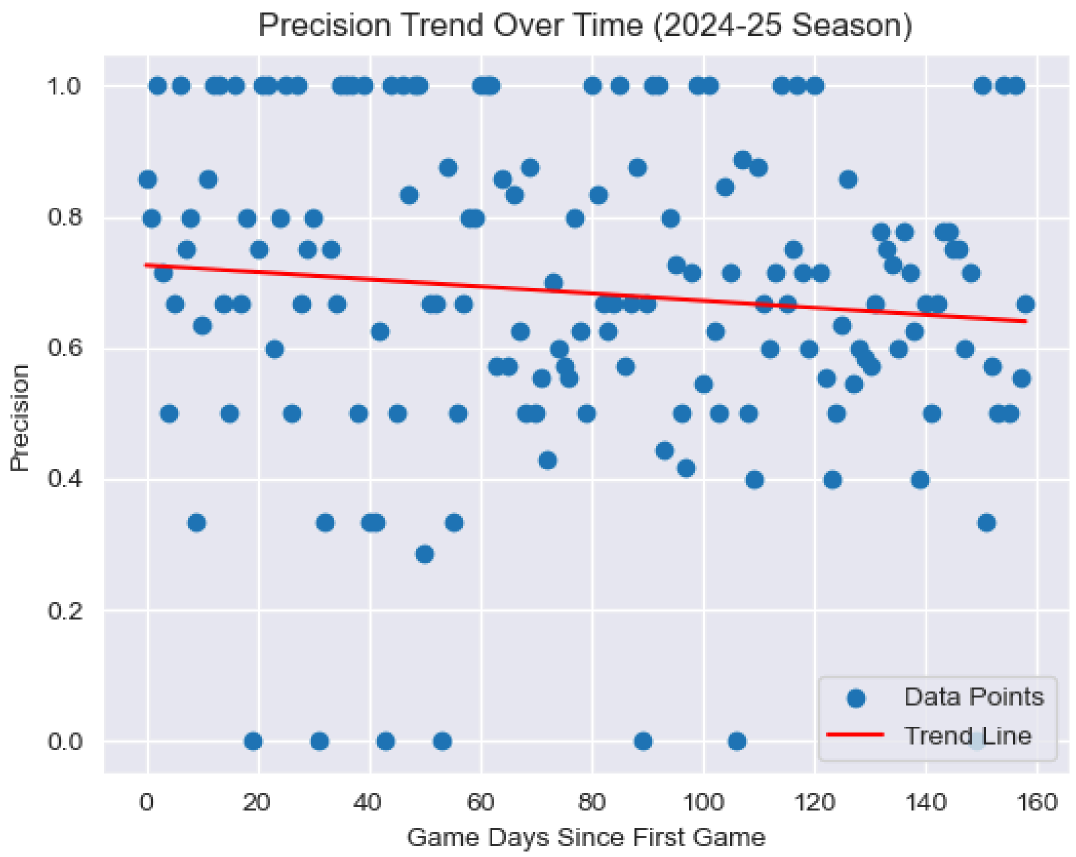
As previously noted, we backtested our model using the 2024–25 NBA season games to gain a clearer understanding of its true predictive performance. The corresponding confusion matrix is shown below.

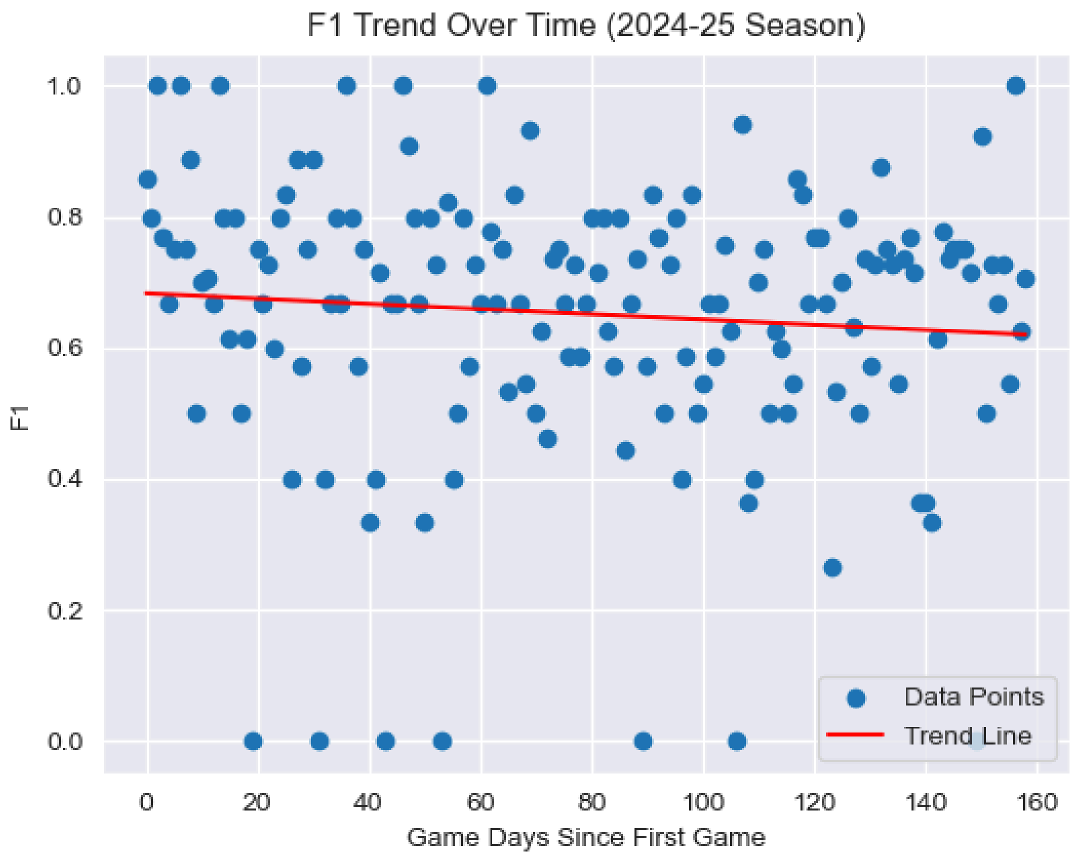


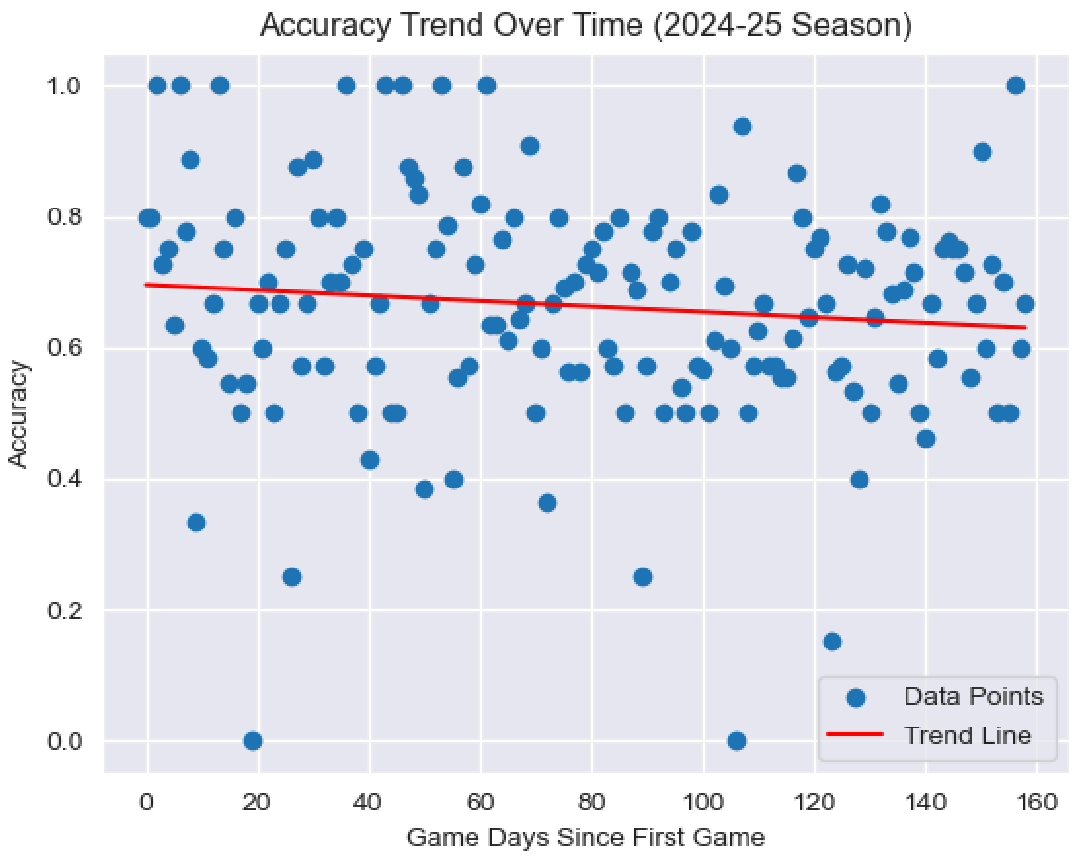
As observed, the confusion matrix shows 620 true positives, 515 true negatives, 272 false positives, and 313 false negatives. This corresponds to a recall of 0.66, precision of 0.70, F1 score of 0.68, and accuracy of 65.99%, which aligns closely with the metrics obtained during training. While our model is not without limitations, the results highlight the fundamental uncertainty and complexity involved in predicting basketball game outcomes within the context of sports betting.

Since the data is time series–based, it is reasonable to expect model performance to degrade as the test window moves further from the training period, due to evolving factors such as team roster changes, player aging, and other dynamic contextual variables. Note: following standard convention, if the denominator in the calculation of recall, precision, or F1 score is zero, the corresponding metric — while technically undefined — is set to 0.


Recall Trend Over Time (2024-25 Season)
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120
140
160
Game Days Since First Game
Data Points
Trend Line


Precision Trend Over Time (2024-25 Season)
Precision
Game Days Since First Game
Data Points
Trend Line
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120
140
160


F1 Trend Over Time (2024-25 Season)
F1
Game Days Since First Game
Data Points
Trend Line
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
120
140
160



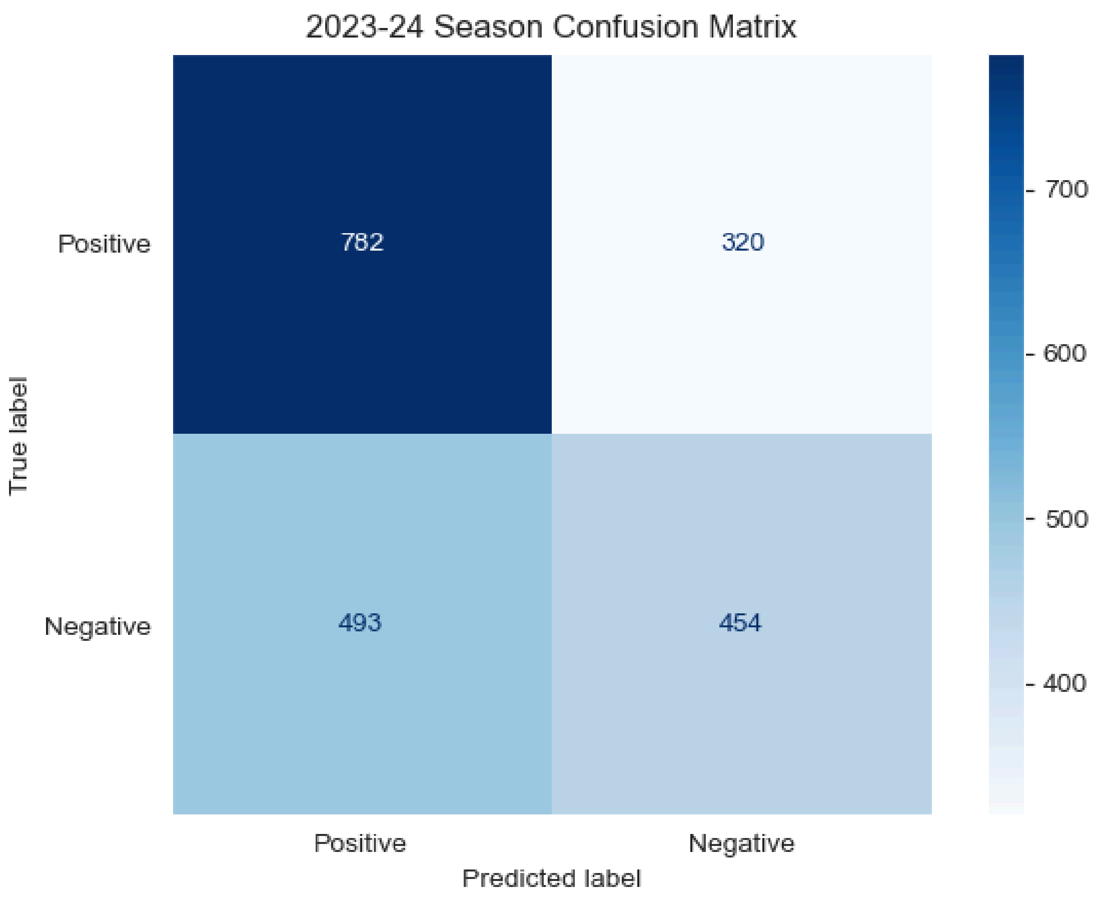
The red trendline in all four graphs exhibits a slight downward trajectory, aligning with our expectation that model performance gradually declines as the temporal distance from the training window increases. Notably, none of the four metrics vary by more than 0.1 over time, with recall demonstrating the least change—successfully achieving our training objective of prioritizing and stabilizing recall performance.

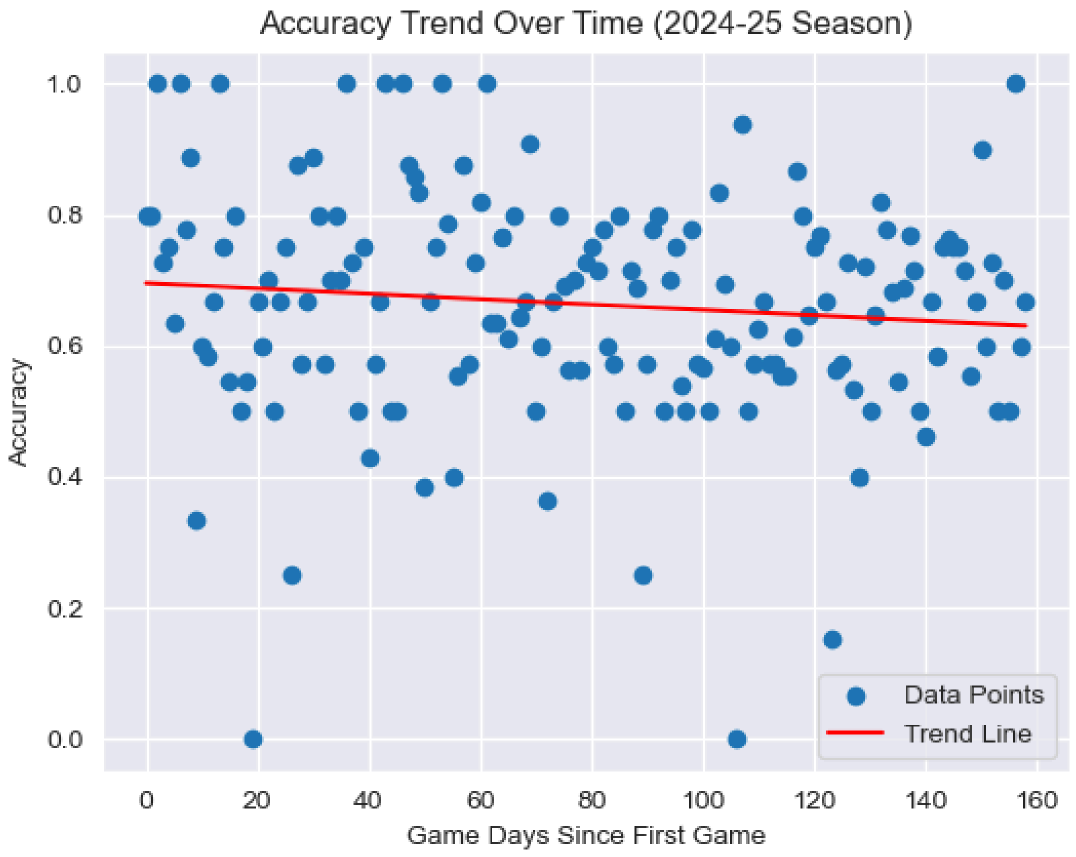
Appendix

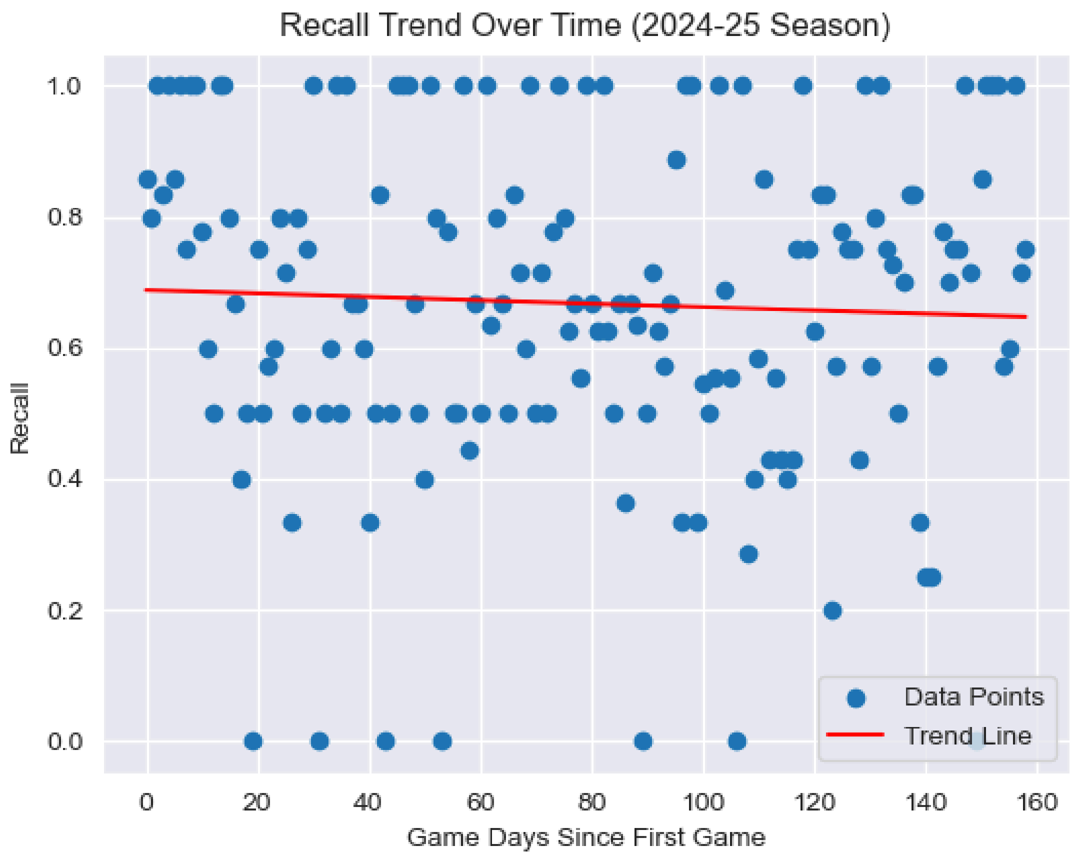
All curated images related to the machine learning (ML) model, including those previously shown, will be compiled and presented here.

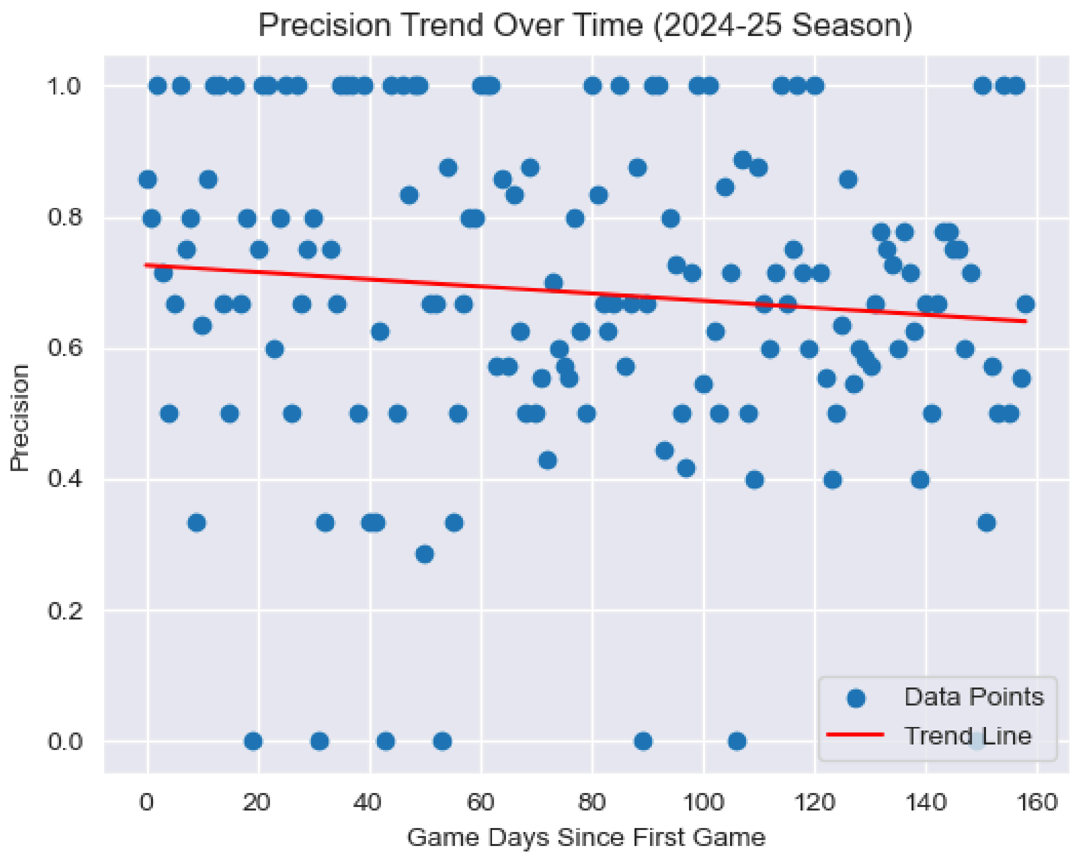
Metrics from testing the ML model on the 2024-25 NBA basketball season.

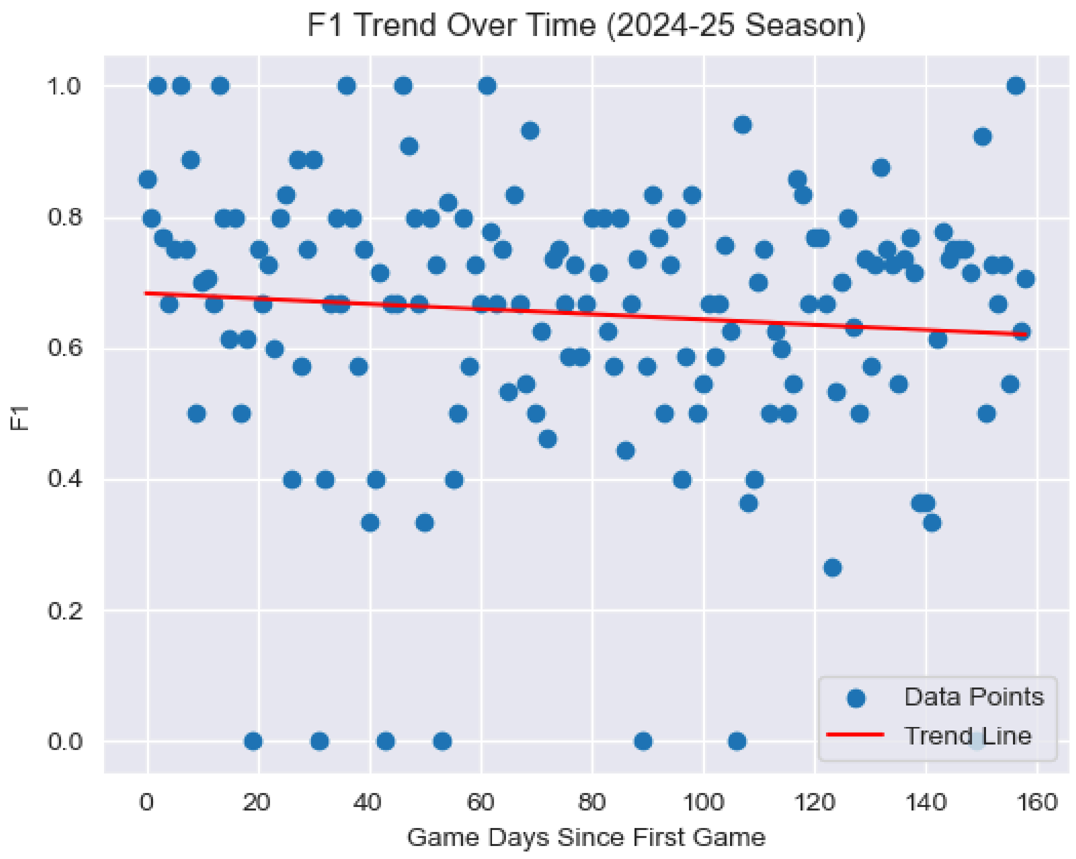
Some of these images below are outdated. Get their most recent version from the repo… the order of them should remain the same


2023-24 Season Confusion Matrix
True label
Positive
Negative
782
320
493
454
Positive
Negative
Predicted label
700
600
500
400


Accuracy Trend Over Time (2024-25 Season)
Accuracy
Game Days Since First Game
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
120
140
160
Data Points
Trend Line


Recall Trend Over Time (2024-25 Season)
Recall
Game Days Since First Game
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
120
140
160
Data Points
Trend Line


Precision Trend Over Time (2024-25 Season)
Precision
Game Days Since First Game
Data Points
Trend Line
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120
140
160



Metrics from the training process for each of the 10 folds for the ML model.

Fold 1:

Correlation matrix before feature selection/reduction:

IMAGE HERE (at backend/models/ml_img/Feature Correlation Matrix (Threshold Magnitude of 0.7) for Pre-Fold 1.png)

Correlation matrix after feature selection/reduction:

IMAGE HERE (at backend/models/ml_img/Feature Correlation Matrix (Threshold Magnitude of 0.7) for Post-Fold 1.png)

Confusion matrix:

IMAGE HERE (at backend/models/ml_img/fold_1_confusion_matrix.png)

Fold 2:

***And so on… continue displaying this pattern for the remaining images until fold 10 is all displayed on the webpage version of this file. ***