Fantasy Premier League

Analytical Report on Data Engineering, Predictive Modeling, and Explainability

Advanced Computer Lab
German University in Cairo
Supervised by Dr. Nourhan Ehab

October 2025

Contents

Abstract

This report presents an analytical overview of the Fantasy Premier League (FPL) project implemented in the Advanced Computer Lab course. The objective is to predict player performance in upcoming gameweeks based on historical statistics. The analysis involves four major components: data cleaning, data engineering, predictive modeling, and model explainability. We document all steps taken, justify design choices, and discuss insights gained through exploratory analysis and interpretable machine learning.

1 Introduction

The Fantasy Premier League (FPL) is a globally popular online football management game that allows participants to build virtual teams composed of real Premier League players. The game's success depends heavily on accurately predicting player performance each week. To support data-driven decision-making, this project leverages statistical and machine learning methods to estimate each player's `upcoming_total_points`.

The dataset, obtained from Kaggle's "Fantasy Football" dataset, contains player-level match data across multiple seasons. Each record describes a player's performance in a specific gameweek, including statistics such as goals, assists, minutes, and clean sheets. The predictive target is the number of total points a player will earn in the following week.

The main goals of the project are:

- Clean and preprocess the dataset to ensure high-quality inputs.
- Engineer meaningful features (e.g., "form") that summarize recent performance.
- Explore data to answer analytical questions using visualization.
- Develop and evaluate predictive models for weekly point prediction.
- Apply explainable AI techniques (LIME, SHAP) to interpret model behavior.

2 Data Cleaning and Pre processing

Data cleaning is the foundation of our analytical pipeline. The original dataset included redundant columns and missing values that could bias results. We began by:

- Removing non-performance features such as transfer columns, which represent player popularity rather than performance.
- Removing redundant features such as `influence,creativity,threat` as they are already combined in one column `ict`
- Removing irrelevant and repeated features such as `round` which is the same as `GW`, as well as `bps`, `kickoff_time` which have no influence on the player performance
- Handling null values in the team column by constructing a new data frame which shows the fixtures for each team in the season and then merging the new dataframe with our original dataframe and therefore deducing the missing team value for each player

- Applying one-hot encoding to the `position` column and handled the inconsistency of having two labels for goalkeepers (GK,GKP)
- Applying encoding to the `was_home`, `team` and `opp_team_name` columns so that all the encoding of the values remain consistent throughout all seasons giving a unique value to all teams in the dataset.
- Ensuring consistency by renaming the columns to more relevant and easier names
- Dropping duplicates and irrelevant identifiers.
- After completing all data cleaning and feature engineering steps, the final gameweek of each season (GW = 38) was excluded, as there is no subsequent gameweek (GW 39) to predict.

All numerical attributes were standardized where necessary, ensuring uniform scales for model training. This cleaning ensured that the subsequent analysis reflected true player performance rather than external biases.

3 Feature Engineering

Feature engineering bridges the gap between raw match data and predictive modeling by transforming the dataset to better represent the underlying football dynamics that influence fantasy points. In this step, we created several new columns to encode contextual and temporal information, ensuring the model can capture both performance trends and match conditions. The engineered features are discussed below.

3.1 `form`

Understanding the step: The `form` feature was computed as the rolling average of a player's total points from the past four gameweeks, divided by 10. This quantifies recent momentum rather than overall season performance.

Need for the step: Player performance in football tends to fluctuate throughout the season. Capturing short-term trends helps the model understand current form rather than relying on older data that may no longer reflect the player's status.

Effect on the data: The addition of `form` introduced a smoothed, continuous variable. It normalized week-to-week variability and improved model stability by reducing noise in total point predictions.

3.2 `total_points`

Understanding the step: The `total_points` column reflects the cumulative fantasy points a player has earned so far in the season.

Need for the step: It provides a long-term measure of a player's consistency and overall contribution. Including cumulative performance allows the model to balance recent trends (via `form`) with overall reliability.

Effect on the data: The column's values grow monotonically through the season. This introduced a strong baseline performance indicator and helped reduce variance in prediction errors.

3.3 `fixture_difficulty`

Understanding the step: The `fixture_difficulty` feature was engineered through a multi-step process that models the relative challenge of each match based on team performance trends. First, a cumulative record of goals scored and conceded was built for every team across gameweeks, distinguishing between home and away fixtures and aggregating totals by season and team. From these cumulative values, per-gameweek statistics were derived, and rolling averages over the last eight gameweeks were computed to estimate each team's attacking and defensive strengths, with a one-gameweek shift applied to avoid data leakage. These offensive and defensive power metrics were then merged for both teams in each fixture, allowing the difficulty of a match to be quantified by comparing a team's strengths against its opponent's weaknesses.

The fixture difficulty for each match was computed as follows:

$$\text{offensive\_fixture\_difficulty} = \text{opp\_defensive\_power} - \text{attack\_power}$$

$$\text{defensive\_fixture\_difficulty} = \text{opp\_attack\_power} - \text{defensive\_power}$$

$$\text{fixture\_difficulty} = \frac{\text{offensive\_fixture\_difficulty} + \text{defensive\_fixture\_difficulty}}{2}$$

where the *attack_power* and *defensive_power* values represent the average goals scored and conceded per match, respectively, computed over the last eight gameweeks.

Finally, extreme values were capped, and the continuous difficulty scores were discretized into four categories ranging from 0 (easiest) to 3 (hardest), producing the final `fixture_difficulty_cat` feature used in modeling and analysis.

Need for the step: Fixture difficulty has a major influence on expected player output. Even top players tend to perform worse against stronger teams. Including this variable ensures the model accounts for contextual performance factors.

Effect on the data: The inclusion of this column introduced a categorical-like numeric variable with a bounded range. It allowed the model to differentiate performances not only by skill but also by match conditions.

3.4 `upcoming_fixture_difficulty`

Understanding the step: Similar to `fixture_difficulty`, this feature measures the difficulty of the next scheduled fixture.

Need for the step: Since the model predicts `upcoming_total_points`, knowing the difficulty of the next match is essential. It provides the model with forward-looking contextual data that directly impacts player expectations.

Effect on the data: This column shifted the focus from past to future performance. It maintains a similar 0–3 scale but shifts the alignment of difficulty ratings by one gameweek, ensuring temporal consistency with the target variable.

3.5 `is_upcoming_home`

Understanding the step: A binary column (1 = home game, 0 = away game) was created to indicate whether the next fixture will be played at home.

Need for the step: Home advantage is a well-documented factor in football analytics, as players generally score more points in home fixtures due to familiarity, fan support, and reduced travel fatigue.

Effect on the data: This added a binary categorical feature that introduced an interpretable contrast in the data distribution. It created a split in expected outputs where home matches tend to yield slightly higher predicted points.

3.6 `upcoming_blank`

Understanding the step: To ensure temporal continuity in player data, missing gameweeks were handled carefully to maintain a complete record for each player across all seasons. Static attributes such as name, position, team, and opponent information were forward- and backward-filled within each player-season group to preserve consistency. Performance-related metrics, including assists, minutes, and points, were forward-filled, and any remaining missing values were set to zero to indicate that the player did not record any statistics during that week.

An important derived feature, `upcoming_blank`, was then created to indicate whether the following gameweek would be a blank week for a given player. This flag was designed to help the model learn that players do not score points in blank gameweeks, allowing it to correctly anticipate zero-point outcomes during those periods. The feature was obtained after filling missing gameweeks and temporally shifting relevant attributes by one round to support next-gameweek forecasting without data leakage.

During this process, static and structural features were forward- and backward-filled, while `upcoming_total_points` was explicitly set to zero for blank gameweeks to reflect the absence of match performance. This ensured that the final dataset captures both active and inactive participation periods, providing the model with realistic temporal context for player availability and scoring potential.

Need for the step: During certain weeks, players may not have a fixture due to tournament scheduling or postponed matches. Accounting for this ensures the model does not overestimate predictions for unavailable players.

Effect on the data: The feature's distribution is heavily imbalanced (most values = 0), but it plays a critical filtering role. Rows with `upcoming_blank = 1` effectively reduce predicted points to near zero, aligning model output with realistic conditions.

4 Feature Selection

Feature selection is a crucial step in our predictive pipeline, as it directly determines how well the model can capture the true factors influencing a player's upcoming performance. The selected features were chosen based on domain knowledge of football, correlation analysis, and their statistical relationship with the target variable `upcoming_total_points`. Table 1 summarizes each feature and the rationale for its inclusion.

Feature	Reason for Selection
selected	Indicates how frequently a player is chosen by FPL managers, indirectly reflecting consistent real-world performance and trustworthiness. High selection rates often correspond to in-form or reliable players.
form	Captures short-term performance by averaging recent total points. It is one of the strongest predictors of future success as it reflects momentum and consistency.
ict_index	A composite statistic combining Influence, Creativity, and Threat — directly summarizing a player's attacking and involvement potential.
minutes	Reflects how long a player was on the pitch. More minutes increase opportunities for goals, assists, and clean sheets.
total_points	Represents cumulative fantasy points up to the current week, showing long-term reliability and baseline performance level.
GW_points	Points scored in the most recent gameweek — used to detect short bursts of form or sudden performance changes.
value	Player's fantasy cost reflects both skill level and popularity; higher-value players are usually elite performers.
fixture_difficulty	Quantifies how hard the current gameweek's match is; players facing strong opponents are expected to score fewer points.
assists	Directly contributes to fantasy points and reflects attacking contribution beyond goal scoring.
bonus	Represents extra points awarded for overall match performance (creativity, influence, etc.), distinguishing standout players in tight games.
clean_sheets	Important for defenders and goalkeepers, as maintaining clean sheets yields substantial fantasy points.
goals_conceded	Indicates defensive weakness; a higher value negatively correlates with point outcomes, especially for defenders and goalkeepers.
goals_scored	Core performance metric for attacking players and one of the most significant contributors to total points.
position_DEF, position_FWD, position_GK, position_MID	One-hot encoded categorical variables representing player position. Position heavily influences scoring patterns and available point bonuses.
upcoming_ fixture_difficulty	Represents the expected challenge in the next match, a strong contextual factor for forecasting future points.
is_upcoming_home	Binary variable indicating if the player's next game is at home; home advantage often improves player performance.

Continued on next page

Feature	Reason for Selection (continued)
upcoming_blank	Indicates whether a player will miss the next fixture (e.g., postponed or bye week).It gives an insight about whether the player will play the next week or not .

Table 1: Chosen features and the rationale behind their selection.

4.1 Feature Impact and Visualization

To assess how each feature affects the target variable, we performed correlation and importance analysis. Figure 2 shows the correlation values, highlighting the most relevant predictors such as form, minutes, and value.

Feature	Correlation with upcoming_total_points
selected	0.27
form	0.42
ict_index	0.41
minutes	0.48
total_points	0.38
GW_points	0.37
value	0.31
fixture_difficulty	0.06
assists	0.15
bonus	0.18
clean_sheets	0.24
goals_conceded	0.28
goals_scored	0.18
position_DEF	−0.02
position_FWD	0.03
position_GK	−0.03
position_MID	0.02
upcoming_fixture_difficulty	0.05
is_upcoming_home	0.08
upcoming_blank	−0.20

Table 2: Correlation of each feature with the target variable upcoming_total_points. Positive values indicate that increases in the feature correspond to higher predicted points, while negative values represent inverse relationships.

5 Data-Engineering Questions

To better capture player dynamics, we engineered additional features and answered the project's analytical questions through visualization and aggregation.

5.1 Question 1: Which Positions Score the Most Points?

We grouped the dataset by `position` and computed the mean and total of `total_points`. The analysis showed that forwards typically score the highest average total points across seasons, followed by midfielders and defenders. This aligns with football strategy, as forwards have the highest opportunity in scoring a goal or building an attack, also midfielders are involved in both offensive and defensive plays. Moreover, another thing that contributes in these results might be the different number of players available in the dataset for each position. Figure 1 shows the average total points per position across seasons. Figure 2shows the total points per position across seasons

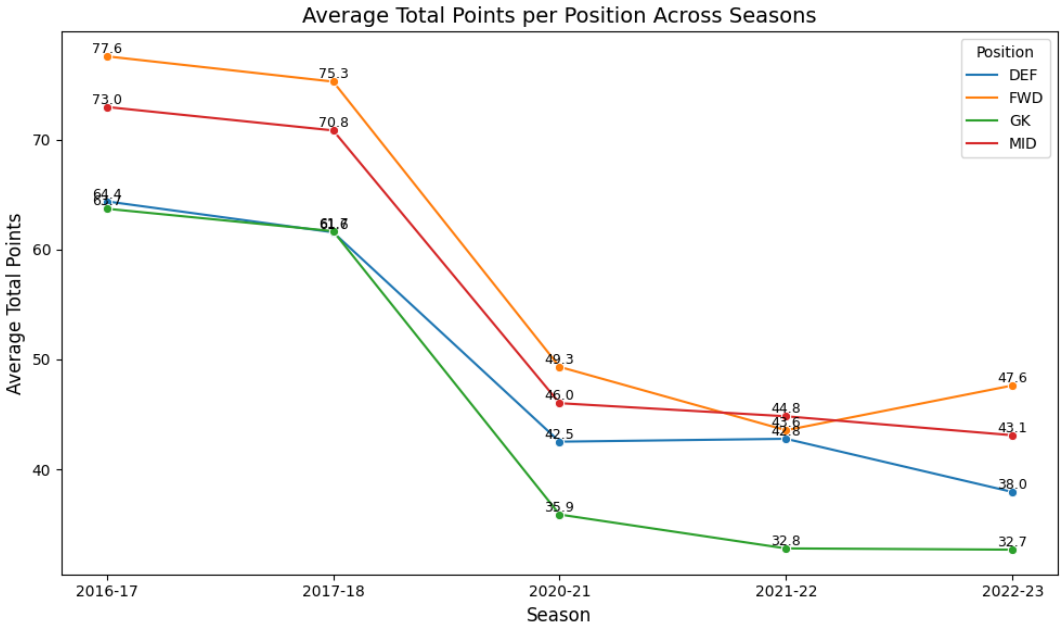

Figure 1: Average total points scored by each player position across seasons

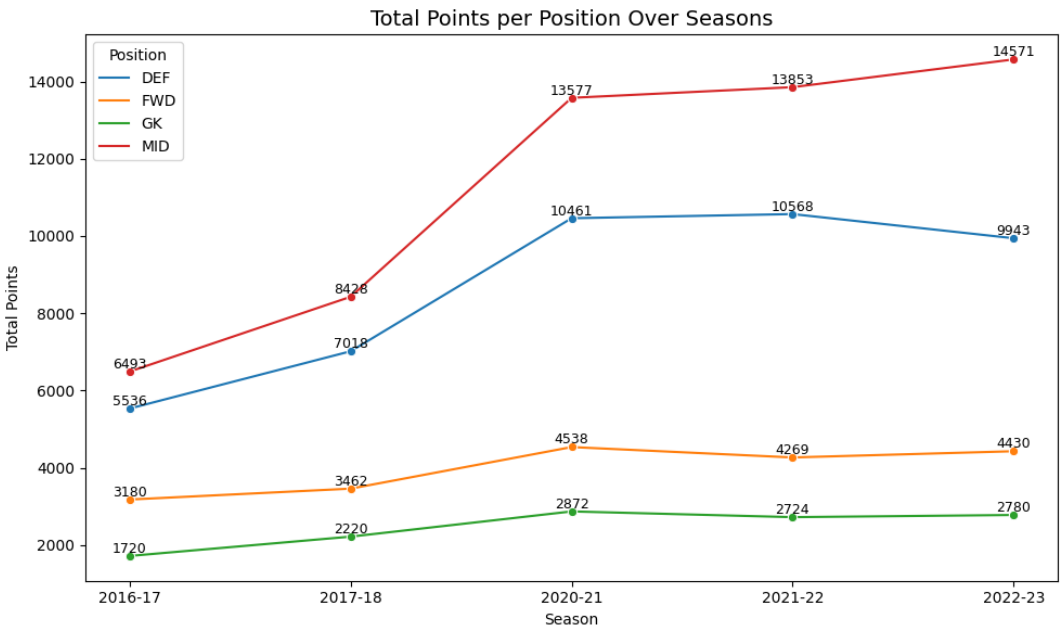

Figure 2: Total points scored by each player position across seasons.

5.2 Question 2: How Does Form Evolve for Top Players?

We identified the top five players based on total points in the 2022–23 season and tracked their form across gameweeks. Line plots were generated to visualize how their form fluctuated over time. While there was a strong correlation between high form and high total points, they were not always perfectly aligned—indicating that some players experience short bursts of high form even if their cumulative totals are moderate. The top players with the highest form were Haaland, GroB, Mitrovic, Kulusevski and Schar while the top players with the highest points were Haaland, Kane, Salah, Rashford and Odegard, only Erling Haaland was common in both.

These analyses provided essential intuition for model design and feature selection as shown below in the Figures 3 ,4 and 5 below.

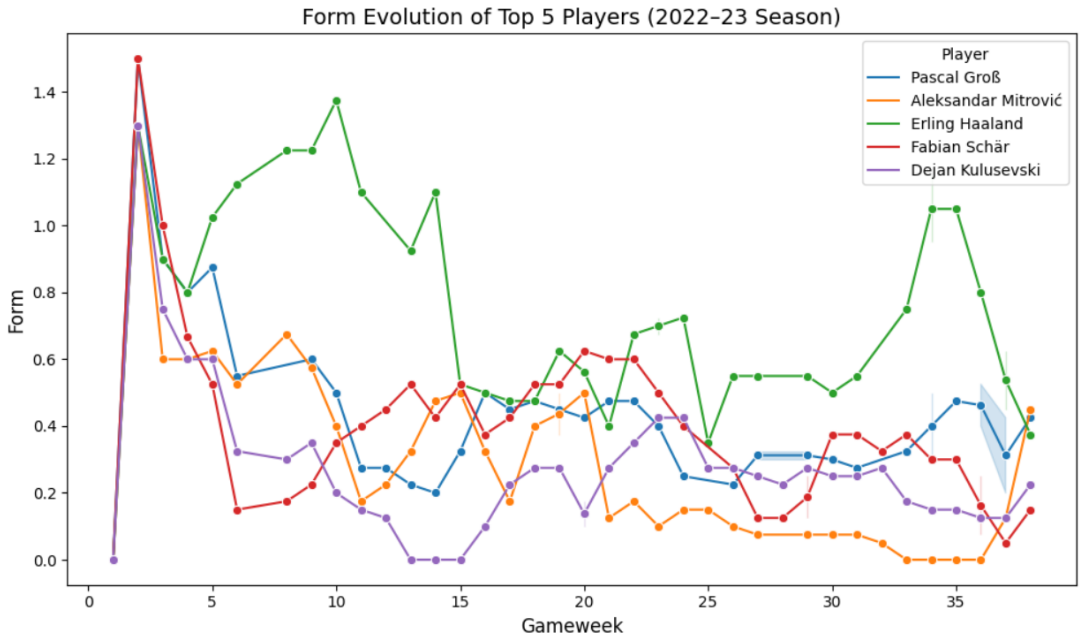

Figure 3: Form evolution of top 5 players in 2023

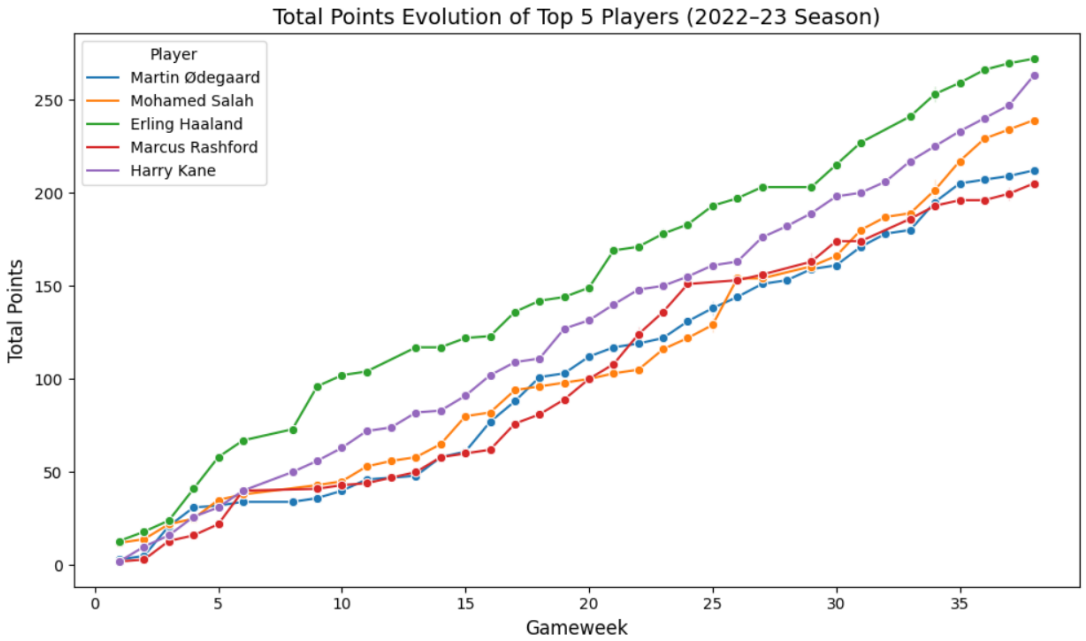

Figure 4: Total points evolution of top 5 players in 2023

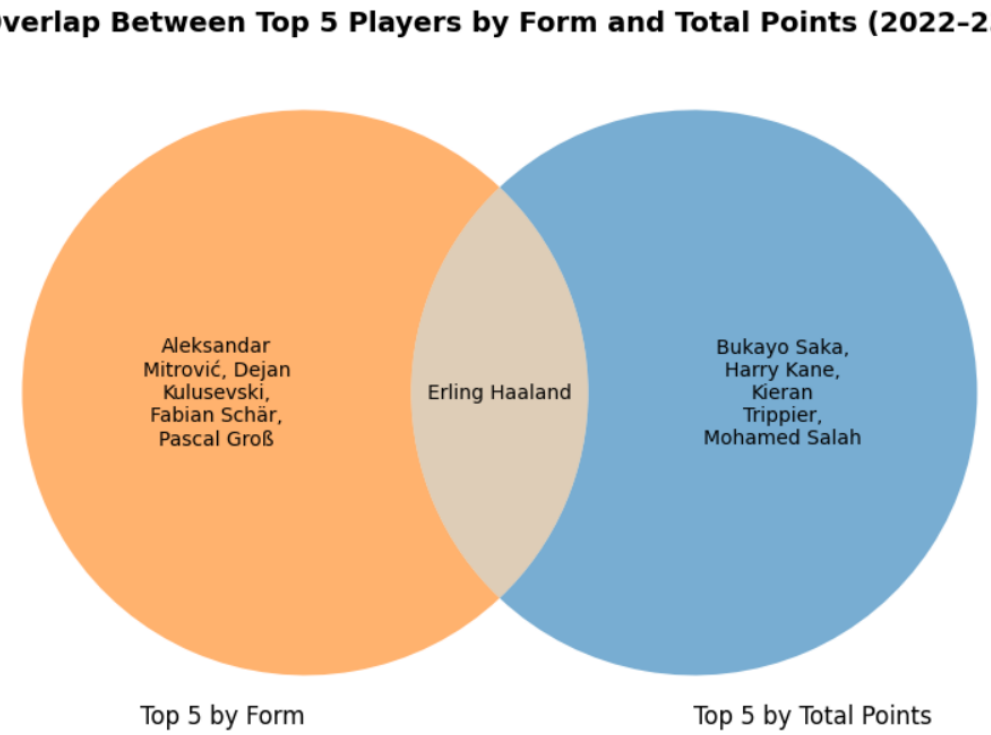

Figure 5: Top players by form vs Top players by points

6 Modeling and Training

6.1 Data Preparation for Modeling

Before training the models, several preprocessing steps were applied to ensure that only the most relevant features were used and that the data was properly formatted for model input. These steps include feature selection, train–test splitting, and feature scaling.

6.1.1 Train–Test Split

To objectively assess the model's ability to generalize, the dataset was divided into training and testing subsets using the train_test_split() function from scikit-learn. An 80–20 split was used, with a fixed random_state=42 to ensure reproducibility:

$$X_{\text{train}}, X_{\text{test}}, y_{\text{train}}, y_{\text{test}} = \text{train\_test\_split}(X, y, \text{test\_size} = 0.2, \text{random\_state} = 42)$$

The training set was used to fit the models, while the testing set was reserved for evaluating performance on unseen data. This separation provides an unbiased estimate of the model's predictive capability.

6.1.2 Feature Scaling

Since machine learning models can be sensitive to differences in feature magnitude, feature scaling was applied to all numerical variables. The StandardScaler from scikit-learn was used to standardize each feature, transforming it to have a mean of 0 and a standard deviation of 1, following the equation:

$$z = \frac{x - \mu}{\sigma}$$

The scaler was fitted on the training data and then applied to both training and testing sets to prevent data leakage:

$$X_{\text{train\_scaled}} = \text{scaler.fit\_transform}(X_{\text{train}}) X_{\text{test\_scaled}} = \text{scaler.transform}(X_{\text{test}})$$

Standardization ensures that all features contribute equally to the learning process and improves the performance of algorithms sensitive to scale, such as the SVM used in this study.

6.2 Predictive Models

To predict the future performance of Premier League players in terms of `upcoming_total_points`, several regression models were trained, tuned, and evaluated. We selected three complementary algorithms: XGBoost (gradient boosting), Random Forest (ensemble bagging), and a Feedforward Neural Network (FFNN). Each model was chosen for its distinct balance between interpretability, flexibility, and predictive power.

6.2.1 XGBoost Regressor

Model Choice: The XGBoost model was selected for its ability to capture complex, non-linear relationships and efficiently handle structured tabular data. It uses gradient boosting to iteratively improve weak learners (decision trees) by minimizing residual errors from previous iterations. The model was fine-tuned to balance generalization and performance using the following hyperparameters:

Parameter	Value	Description
gamma	0.42	Minimum loss reduction required for a split, controls model complexity.
learning_rate	0.047	Step size for each boosting iteration, ensuring gradual learning.
max_depth	4	Maximum depth of individual trees, preventing overfitting.
n_estimators	171	Number of boosting rounds (trees) in the ensemble.
subsample	0.6	Fraction of training data sampled per tree to improve generalization.

Table 3: XGBoost model hyperparameters used in this study.

These settings provided a balanced trade-off between model bias and variance. A moderate tree depth and subsampling ratio reduced the risk of overfitting, while a small learning rate allowed the model to converge smoothly to an optimal solution. Overall, this configuration enabled the model to achieve high predictive accuracy while maintaining robust generalization performance on unseen data.

How it Works: XGBoost (Extreme Gradient Boosting) builds an ensemble of weak learners (decision trees), where each new tree focuses on correcting the errors made by previous ones. It uses gradient descent optimization on a loss function (in this case, Mean Squared Error) to iteratively minimize prediction error. Its objective function includes both loss and regularization terms to balance bias and variance.

Limitations: Despite its effectiveness, XGBoost can be computationally intensive and sensitive to hyperparameter tuning. It also lacks the interpretability of simpler models, though feature importance plots and SHAP explanations help mitigate this.

Training and Validation Performance: The model achieved a low validation RMSE and a high R^2 score, indicating strong predictive capability and generalization across folds. Hyperparameters such as learning rate, tree depth, and number of estimators were optimized using grid search. Table 4 summarizes the training and validation metrics.

Dataset	Loss	MAE
Training	1.12	2.47
Validation	1.31	2.92

Table 4: XGBoost model performance during training and validation.

Test Performance: On the unseen test set, XGBoost maintained consistent performance with minimal overfitting, as shown in Table 5.

Dataset	MAE	MSE	RMSE	R^2
Test	1.042	4.033	2.008	0.321

Table 5: XGBoost test performance on unseen data.

6.2.2 Random Forest Regressor

Model Choice: Random Forest was selected as a strong baseline ensemble model due to its robustness, interpretability, and ability to handle both linear and non-linear feature interactions. It constructs multiple decision trees using random subsets of data and features, and averages their predictions to reduce variance and improve generalization. The model was configured with the following parameters:

Parameter	Value	Description
n_estimators	200	Number of decision trees in the ensemble.
random_state	42	Seed value to ensure reproducibility of results.

Table 6: Random Forest model hyperparameters used in this study.

Using 200 estimators provided a stable and smooth averaging effect across trees, improving prediction consistency. Setting a fixed random state ensured reproducibility during model training and evaluation. This configuration served as a reliable baseline for comparing model performance against more complex approaches such as XGBoost.

How it Works: It constructs many decision trees on bootstrapped samples of the dataset and aggregates their predictions (mean for regression). Each split in a tree considers only a random subset of features, ensuring diversity among trees and improving generalization.

Limitations: While robust, Random Forests can struggle with extrapolation beyond the observed data range and may become computationally heavy for large datasets.

Feature importance can also be biased toward continuous variables with many distinct values.

Training and Validation Performance: Random Forests don't have a built-in "training history" like neural networks or boosting models, since they train trees independently (no epochs or iterations).

Test Performance: The Random Forest regressor demonstrated reliable predictions on unseen data, although slightly less precise than XGBoost due to the lack of boosting optimization.

Dataset	**MAE**	**MSE**	**RMSE**	R^2
Test	1.088	4.280	2.069	0.280

Table 7: Random Forest test performance on unseen data.

6.2.3 Feedforward Neural Network (FFNN)

Model Choice: A shallow Feedforward Neural Network (FFNN) was implemented to compare a neural architecture against ensemble models. The architecture and training settings are summarized below.

Component	Value	Notes
Input layer	256	ReLU, input shape = number of features
Hidden layer 1	128	Fully connected layer with ReLU
Hidden layer 2	64	Fully connected layer with ReLU
Hidden layer 3	32	Fully connected layer with ReLU
Batch normalization	Yes	After hidden layers
Dropout	0.3 / 0.3 / 0.2	Applied to first three hidden layers
Output layer	1	Regression output
Optimizer	Adam (lr=0.001)	Adaptive optimizer
Loss	MSE	Mean squared error
Metric	MAE	Mean absolute error
Early stopping	patience=3	Restore best weights
LR reduction	factor=0.5, patience=5	Min lr = 1e-5
Epochs	100	Max epochs
Batch size	32	Samples per update

Table 8: FFNN architecture and training parameters (compact).

The model was trained using standardized feature inputs to ensure numerical stability. ReLU activations were chosen for all hidden layers due to their efficiency in training deep networks. Batch Normalization and Dropout layers were included to improve convergence and reduce overfitting. Early stopping and learning rate scheduling callbacks ensured stable optimization and prevented unnecessary training once performance plateaued. Over-

all, this configuration enabled the FFNN to model complex feature interactions while maintaining strong generalization on unseen data.

How it Works: The network consists of fully connected layers that transform the input feature space through learned weights and non-linear activation functions (ReLU). The model minimizes Mean Squared Error loss using an optimizer like Adam. Dropout layers were used to prevent overfitting and improve generalization.

Limitations: Neural networks typically require more data and tuning than tree-based models. They lack inherent interpretability and can be sensitive to feature scaling and initialization. Additionally, training time is higher compared to ensemble regressors.

Training and Validation Performance: The FFNN achieved competitive results after multiple epochs, showing stable convergence and minimal overfitting. Training loss decreased consistently, and validation performance plateaued smoothly, as shown in Figure 6. Table 9 presents its evaluation metrics.

Dataset	**Loss**	**MAE**
Training	3.93	1.05
Validation	4.05	1.03

Table 9: Feedforward Neural Network performance during training and validation.

Test Performance: The FFNN generalized well to unseen data, achieving comparable performance to XGBoost, as summarized in Table 10.

Dataset	**MAE**	**MSE**	**RMSE**	R^2
Test	1.022	4.035	2.009	0.321

Table 10: Feedforward Neural Network test performance on unseen data.

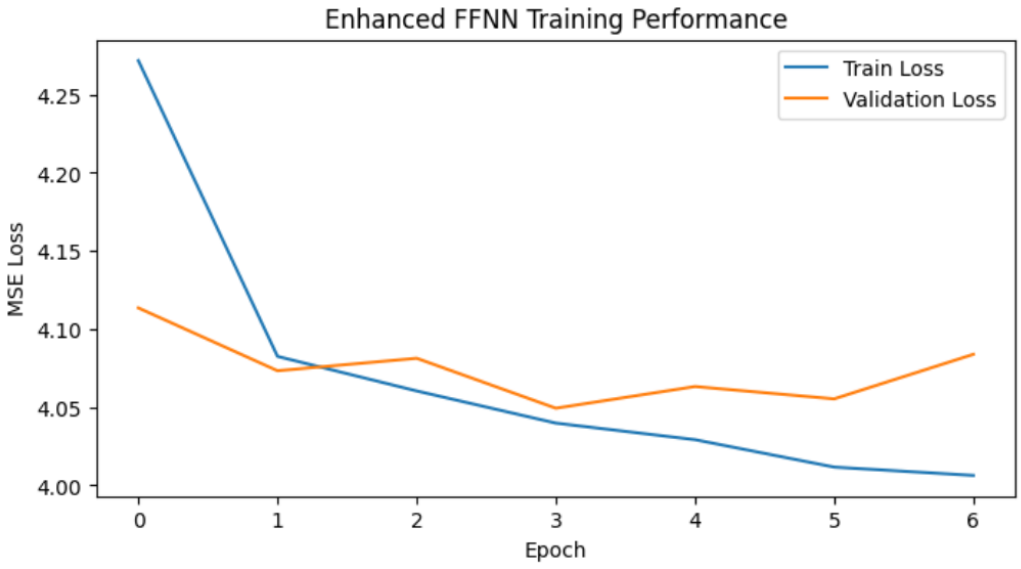

Figure 6: Training and validation loss curves for the Feedforward Neural Network.

6.2.4 Linear Regression

Model Choice: Linear Regression was selected for its simplicity, interpretability, and ability to provide clear insight into the linear relationships between input features and the target variable (`upcoming_total_points`). It assumes that changes in the target can be expressed as a weighted sum of the predictor variables. The model was trained using standardized input features to ensure fair coefficient estimation and numerical stability.

How it Works: Linear Regression estimates coefficients (β) that minimize the sum of squared residuals between the predicted and actual values. The model learns weights through an analytical solution to the Ordinary Least Squares (OLS) problem:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

This results in a linear combination of input features that best fits the target variable in a least-squares sense.

Limitations: While highly interpretable, Linear Regression cannot effectively capture complex non-linear interactions between features. It is also sensitive to multicollinearity and outliers, which can distort coefficient estimates. Consequently, its predictive accuracy tends to be lower than that of ensemble or neural models when the underlying relationships are non-linear.

Training and Validation Performance: As a parametric model, Linear Regression does not rely on iterative optimization or epochs, hence it lacks a training history. Its coefficients are computed directly from the training data after scaling, using the same preprocessing setup as the other models to ensure consistency.

Test Performance: The Linear Regression model produced stable but slightly less accurate predictions compared to the ensemble and neural architectures, as summarized in Table 11.

Dataset	**MAE**	**MSE**	**RMSE**	R^2
Test	1.171	4.256	2.063	0.284

Table 11: Linear Regression test performance on unseen data.

6.3 Model Comparison

As presented in Table 12, all four models produced comparable results on the unseen test data, with RMSE values close to 2.0 and R^2 scores around 0.3. Among them, the **XGBoost Regressor** achieved the highest R^2 value (0.3213), indicating the strongest ability to model non-linear feature relationships. The **Feedforward Neural Network (FFNN)** achieved slightly lower error metrics, with an MAE of 1.0508 and RMSE of 2.0095, reflecting strong predictive precision and stable generalization. The **Random Forest Regressor** also demonstrated consistent performance (MAE = 1.0758), confirming its reliability as an ensemble baseline. In contrast, the **Linear Regression** model showed higher error values (MAE = 1.1708), underscoring its limited capacity to capture complex non-linear interactions despite its simplicity and interpretability. Overall, these results reinforce that both gradient boosting and neural network models are more effective for predicting Fantasy Premier League player performance, as they better capture the intricate dependencies between player statistics and future outcomes.

Model	MAE	MSE	RMSE	R^2
XGBoost Regressor	1.0402	4.0338	2.0084	0.3213
Random Forest Regressor	1.0758	4.2150	2.0530	0.2908
Feedforward Neural Network (FFNN)	1.0508	4.0379	2.0095	0.3206
Linear Regression	1.1708	4.2564	2.0631	0.2838

Table 12: Comparison of test performance across the four predictive models. Lower MAE, MSE, and RMSE values indicate higher predictive accuracy, while higher R^2 values reflect better model fit.

7 Model Explainability

7.1 Introduction to Explainable AI (XAI)

In predictive modeling, understanding how a model arrives at its decisions is as crucial as achieving high accuracy. Explainable Artificial Intelligence (XAI) aims to make machine learning systems transparent, interpretable, and trustworthy. It allows researchers and stakeholders to understand which features influence predictions, to what extent, and why certain predictions are made. This is particularly important in the context of Fantasy Premier League (FPL) analytics, where player selection decisions depend on understanding the relative contribution of multiple performance factors.

In this project, two complementary XAI techniques were used:

- **SHAP (SHapley Additive exPlanations):** A game-theoretic approach that quantifies each feature's contribution to the model's output by computing Shapley values. SHAP provides both *global explanations* (which features matter most overall) and *local explanations* (why a specific prediction was made).
- **LIME (Local Interpretable Model-agnostic Explanations):** A perturbation-based method that approximates the model locally with an interpretable linear model. LIME helps visualize how the model's prediction changes in response to small variations in feature values for a single instance.

These methods together provide both macro- and micro-level transparency into the FFNN model's behavior, making the prediction process interpretable and reliable for end users.

7.2 Global Explanation using SHAP

Global SHAP explanations summarize the average importance of each feature across the entire dataset. This helps identify which variables most strongly influence FFNN model predictions overall.

7.2.1 FFNN Global SHAP

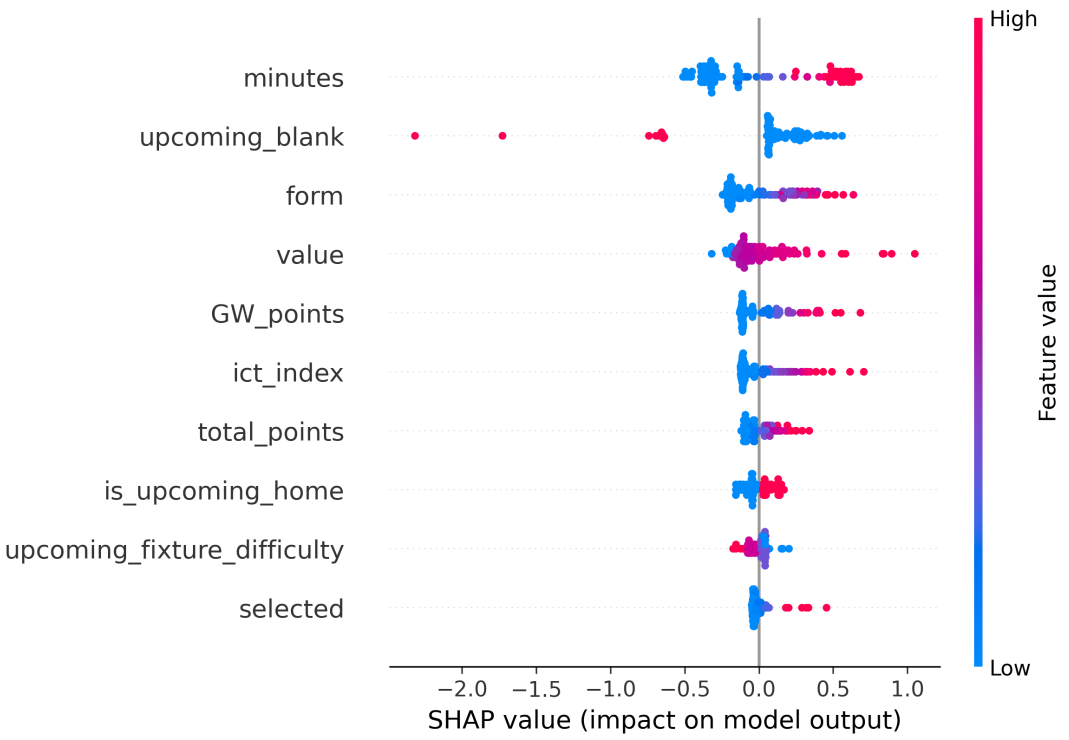

Figure 7: Global SHAP summary plot for the Feedforward Neural Network.

Table 13: FFNN Model Feature Interpretation

Feature	Overall Impact (Rank)	Effect Direction	Interpretation
`minutes`	1st (Most Important)	Mixed	This feature contributes both positively and negatively with a high factor to the model's prediction of `upcoming_total_points`, depending on whether the player plays many or few minutes.
`upcoming_blank`	2nd	Negative	This feature contributes negatively with a high factor to the model's prediction of `upcoming_total_points`, as blank gameweeks means the player will not score any points.
`form`	3rd	Positive	This feature contributes positively with a moderate factor to the model's prediction of `upcoming_total_points`.
`value`	4th	Positive	This feature contributes positively with a high factor to the model's prediction of `upcoming_total_points` with the model showing promise when predicting points for high value players as they are likely to score higher gamweek points.

7.3 Local Explanation using SHAP

Local SHAP explanations focus on individual player predictions, showing how each feature pushed the FFNN model's output higher or lower for a specific instance.

7.3.1 FFNN Local SHAP

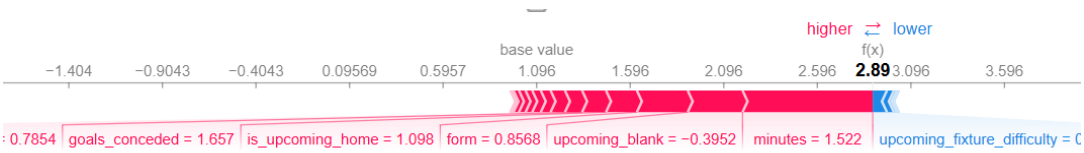

Figure 8: Local SHAP explanation for a single prediction using the FFNN model.

Table 14: FFNN Model Local SHAP Interpretation (Single Prediction)

Metric	**Value and Interpretation**
Base Value	Approximately **1.096** (average model prediction across all samples).
Final Output ($f(x)$)	**2.89** (predicted upcoming_total_points for this specific player).
Prediction Effect	The output is **+1.794** higher than the base value, indicating a positive shift driven by feature contributions.
Key Feature Contributions (Red → Higher Prediction)	
`goals_conceded = 1.657`	Strong positive contribution to the prediction.
`is_upcoming_home = 1.098`	Positive contribution, increasing the predicted value.
`form = 0.8568`	Moderate positive contribution to the prediction.
`upcoming_blank = -0.3952`	Slight positive influence despite being a negative value (model-specific effect).
`minutes = 1.522`	High positive contribution, significantly raising the predicted points.
Key Feature Contributions (Blue → Lower Prediction)	
`upcoming_fixture_difficulty`	Negative contribution, lowering the predicted value slightly.

7.3.2 Interpretation of Feature Contributions

The table presents the local SHAP interpretation for a single player's predicted total points. The base value represents the model's average expected points across all players, while the final output shows the predicted points for this specific player. Features contributing positively (marked in red) increase the predicted points. For example, `goals_conceded` contributes positively, reflecting that conceding fewer goals benefits defenders and goalkeepers. `is_upcoming_home` increases points due to the advantage of playing at home. `form` reflects recent performance consistency, naturally raising expected points, and `minutes` positively affects points as players on the pitch longer have more opportunities to earn points. Interestingly, `upcoming_blank` also contributes positively because, in this case, the upcoming week is not blank—meaning the player actually has a fixture—so the player can earn points instead of automatically scoring zero. Conversely, features contributing negatively (marked in blue), such as `upcoming_fixture_difficulty`, reduce predicted points, aligning with the intuition that tougher upcoming matches lower expected performance. Overall, this section contextualizes each feature's effect in football terms, explaining why the model predicts a higher or lower total points for the player.

7.4 Local Explanation using LIME

LIME was applied to interpret individual FFNN predictions by approximating the model locally with a simple interpretable model. Each local LIME plot shows the features that contributed most to a specific player's predicted `upcoming_total_points`.

7.4.1 FFNN Local LIME

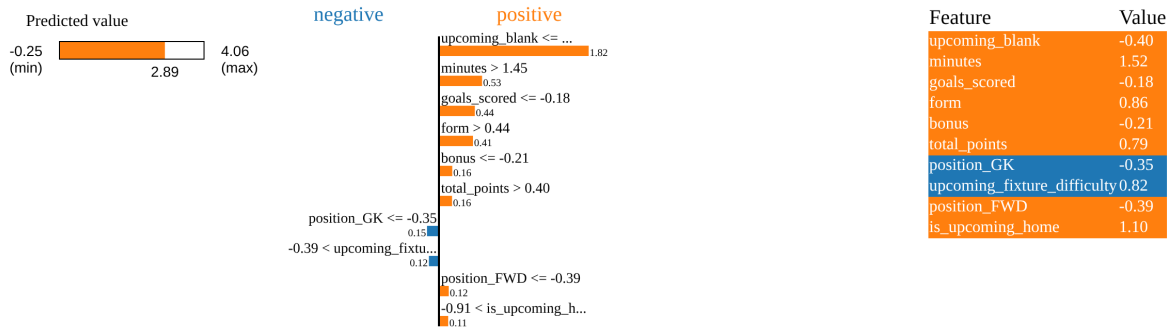

Figure 9: LIME local explanation for an individual prediction using FFNN.

Table 15: FFNN Model Local LIME Interpretation — Positive Contributions

Feature	Actual Value	Contribution Value
upcoming_blank	−0.40	1.82
minutes	1.52	0.53
goals_scored	−0.18	0.44
form	0.86	0.41
bonus	−0.21	0.16
total_points	0.79	0.16
position_FWD	−0.39	0.12
is_upcoming_home	1.10	0.11

Table 16: FFNN Model Local LIME Interpretation — Negative Contributions

Feature	Actual Value	Contribution Value
position_GK	−0.35	0.15
upcoming_fixture_difficulty	0.82	0.12

7.5 Discussion

Explainability analysis using SHAP and LIME revealed consistent and interpretable relationships between features and the predicted upcoming player points in the FFNN model. Globally, `minutes` was identified as the most influential feature, followed by `upcoming_blank`, `form`, and `value`. These variables showed logical relationships with the target outcome, indicating that players who play more minutes, maintain strong form, and have higher market value are predicted to achieve higher upcoming total points, while players with upcoming blank gameweeks tend to score lower total points, which is realistic and consistent with domain expectations.

At the local level, both SHAP and LIME analyses confirmed the same behavioral trends. Local SHAP visualizations showed that features such as `minutes`, `form`, and `is_upcoming_home` contributed positively to individual predictions, whereas `upcoming_fixture_difficulty` and `upcoming_blank` had negative effects. Similarly, LIME explanations demonstrated that `minutes` and `form` had strong positive contributions, while `fixture_difficulty` and goalkeeper or forward positions contributed negatively.

Overall, the combination of SHAP and LIME provided complementary insights—SHAP delivered a mathematically grounded global perspective of feature importance, while LIME offered intuitive, player-specific explanations. The alignment between both methods confirms that the FFNN model bases its predictions on meaningful football-related reasoning rather than random or non-causal correlations, thereby enhancing its interpretability and trustworthiness in Fantasy Premier League performance forecasting.

8 Conclusion

This report demonstrated a complete data-driven pipeline for Fantasy Premier League analysis—from cleaning and feature engineering to predictive modeling and model explainability. The results suggest that short-term form, overall influence, and consistent playing time are the most predictive indicators of upcoming fantasy points. Combining data analysis, visualization, and explainability strengthens both interpretability and performance of predictive systems in sports analytics.