INTERNSHIP REPORT :

ORGANISATION-INFOSYS

TITLE- PLAYER TRANSFER VALUES PREDICTIONS

NAME- SHRABANI MONDAL

STUDENT ID- 2301287466

TECHNOLOGY TRACK- AI

SUBMISSION DATE- 02.01.2026

Introduction

Player transfer markets constitute a critical component of professional sports, significantly influencing team composition, financial strategies, and overall competitive performance. Accurately estimating a player's transfer value is a complex task, as it depends on multiple interrelated factors such as on-field performance, age, club affiliation, public sentiment, injury history, and prevailing market trends.

This project proposes the development of an AI-driven model for predicting player transfer values by integrating data from multiple sources, including player performance statistics, social media sentiment analysis, injury records, and historical market value information. By employing advanced machine learning algorithms along with time-series forecasting techniques, the model aims to capture both player-specific attributes and temporal market dynamics. The proposed approach offers a dynamic, data-driven solution for estimating transfer values, enabling more informed and objective decision-making within the football transfer ecosystem.

Methodology

The project is structured into multiple stages, each focusing on a specific aspect of data processing and model development. The methodology emphasizes the integration of multi-source data to build a comprehensive representation of player profiles and market behavior.

Initially, data collection and preprocessing are performed to clean, transform, and standardize inputs from diverse sources. Feature engineering and selection techniques are then applied to identify the most influential factors affecting transfer values. Subsequently, machine learning models and time-series forecasting methods are employed to learn historical trends and predict future transfer values. Model evaluation and optimization are conducted to ensure robustness and accuracy. This staged approach ensures a systematic and scalable framework for predicting player transfer values using AI-driven techniques.

Data collection:

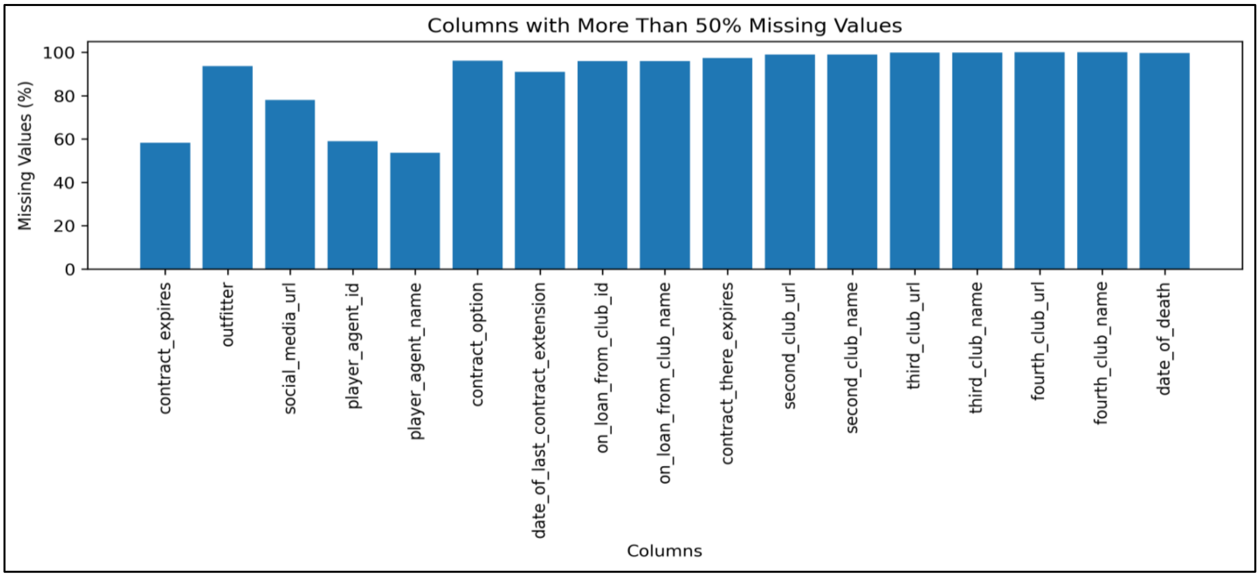
The data used in this study was collected from publicly available datasets (Kaggle) related to professional football players and their performance statistics. Additional data related to player sentiment was obtained from social media sources (Twitter), containing tweets associated with football players. The data was provided in CSV format, which was imported and processed using Python and the Pandas library. The data files include player profile, player performances, market value, player injuries, and sentiment data.

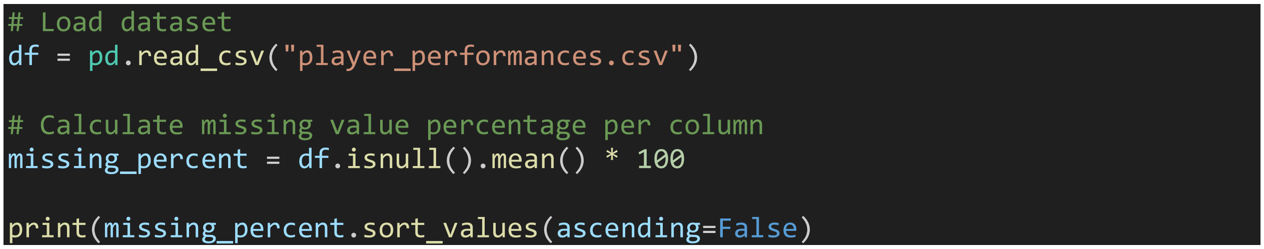
Data Understanding:

Each data file was studied individually to find the relation among them. The dataset includes performance-related attributes such as goals, assists, minutes played, yellow cards, and clean sheets. Player-specific attributes include age, height, nationality, and market value. Injury-related features such as days missed and games missed indicate player availability. Sentiment features like vader polarity and tb polarity represent public opinion extracted from social media tweets.

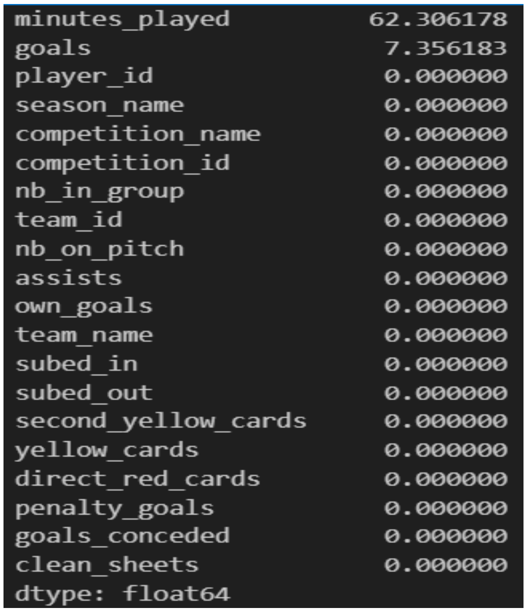
The data set contains numerical columns like goals, assists, yellow cards, clean sheets etc , categorical columns like foot, vader emotion, tb emotion, position etc, text columns like sentiment text , date columns like date-of-birth, joined, contract expires etc.

The player profile data consists of multiple columns which were having high missing values.





```
# Load dataset
df = pd.read_csv("player_performances.csv")

# Calculate missing value percentage per column
missing_percent = df.isnull().mean() * 100

print(missing_percent.sort_values(ascending=False)
```



```
minutes_played          62.306178
goals                    7.356183
player_id                0.000000
season_name              0.000000
competition_name         0.000000
competition_id           0.000000
nb_in_group              0.000000
team_id                  0.000000
nb_on_pitch              0.000000
assists                  0.000000
own_goals                0.000000
team_name                0.000000
subed_in                 0.000000
subed_out                0.000000
second_yellow_cards      0.000000
yellow_cards             0.000000
direct_red_cards         0.000000
penalty_goals            0.000000
goals_conceded           0.000000
clean_sheets             0.000000
dtype: float64
```

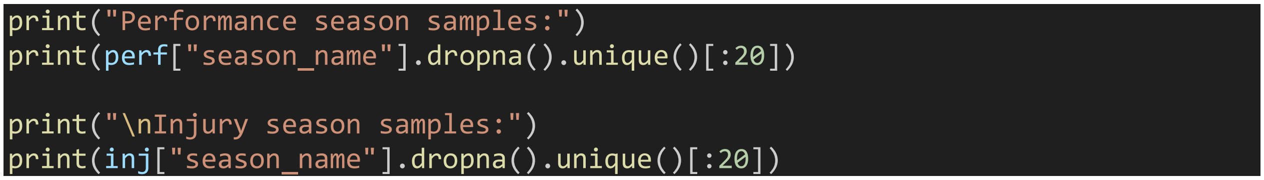
Initial inspection of the dataset revealed a wide variation in player performance metrics.

Certain players showed high minutes played, while others had frequent injuries resulting in high days missed.

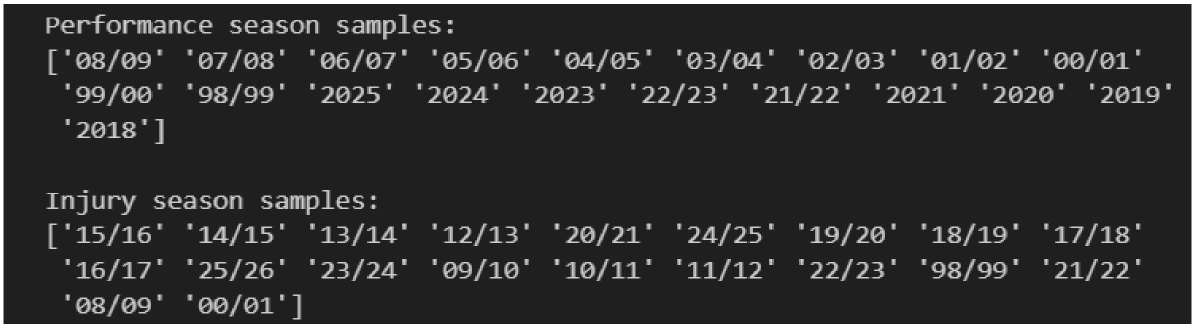
During data exploration, several quality issues were identified, including missing values in injury-related fields, inconsistent data types in count-based columns, and the presence of outliers in features such as minutes played and market value etc . Some columns also contained decimal values where integers were expected. Ex- Goals, days missed, should be having integer as the data type but it was found to have float, which is not possible and these values must be in whole.

The target variable for this study is the player's market value, which is influenced by performance, availability, and sentiment-related features.

There was a variation on the season format was found in player performances and player injury leading to inconsistencies.

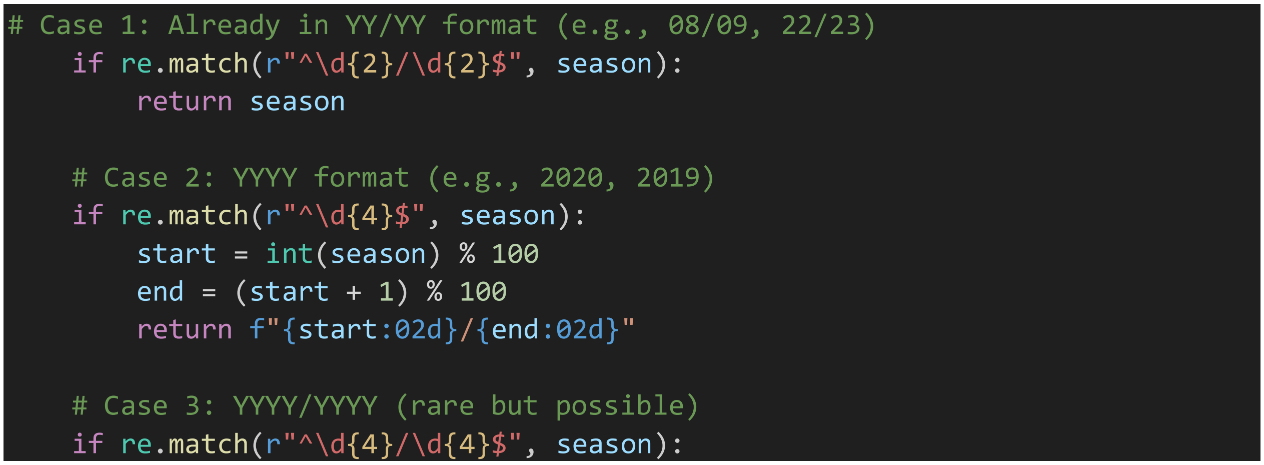


```
print("Performance season samples:")
print(perf["season_name"].dropna().unique()[:20])

print("\nInjury season samples:")
print(inj["season_name"].dropna().unique()[:20])
```

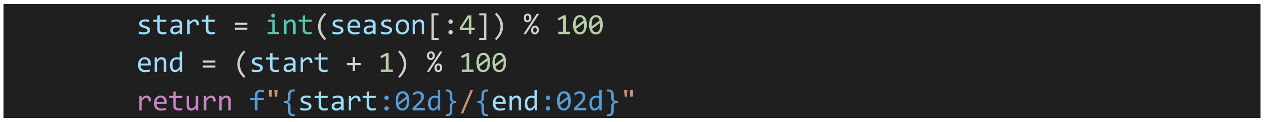


```
Performance season samples:
['08/09' '07/08' '06/07' '05/06' '04/05' '03/04' '02/03' '01/02' '00/01'
 '99/00' '98/99' '2025' '2024' '2023' '22/23' '21/22' '2021' '2020' '2019'
 '2018']

Injury season samples:
['15/16' '14/15' '13/14' '12/13' '20/21' '24/25' '19/20' '18/19' '17/18'
 '16/17' '25/26' '23/24' '09/10' '10/11' '11/12' '22/23' '98/99' '21/22'
 '08/09' '00/01']
```

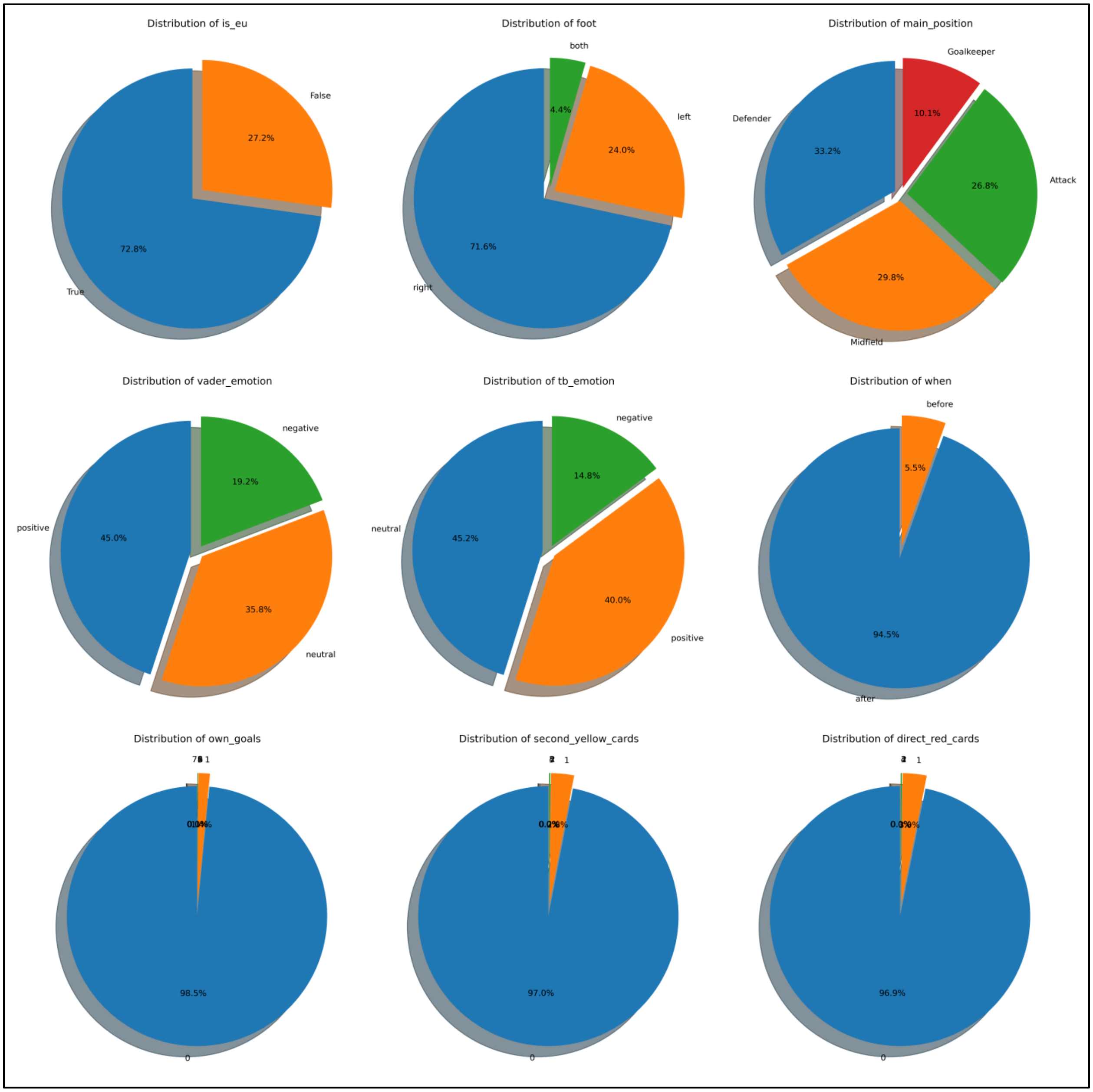
After this analysis the seasons were converted to a standard format for better analysis.



```
# Case 1: Already in YY/YY format (e.g., 08/09, 22/23)
    if re.match(r"^\d{2}/\d{2}$", season):
        return season

    # Case 2: YYYY format (e.g., 2020, 2019)
    if re.match(r"^\d{4}$", season):
        start = int(season) % 100
        end = (start + 1) % 100
        return f"{start:02d}/{end:02d}"

    # Case 3: YYYY/YYYY (rare but possible)
    if re.match(r"^\d{4}/\d{4}$", season):
```



```
    start = int(season[:4]) % 100
    end = (start + 1) % 100
    return f"{start:02d}/{end:02d}"
```



Pie charts were used to analyse the distributions among the columns having lesser distinct values respectively. Features like is eu held mostly true whereas most players use right foot whereas some plays on both foot too. It also shows that majority of the player is having lesser value for cards. While very few have high. However the distribution plot for main position showed equal spread among the categories. The sentiment features like vader emotion and tb emotion showed that most of the player have positive and neutral as the sentiment respectively. This analysis provided us a easy guide for the observation of different features.

Data Merging:

The data was obtained from multiple sources. Data merging was performed to integrate these datasets into a single unified dataset, enabling comprehensive analysis of player performance along with public sentiment.

The datasets were merged using a common identifier, such as player ID and season name, ensuring accurate alignment between performance records and corresponding injury data , market value data etc. A left join operation was applied, using the player performance dataset as the base table. This approach ensured that all player performance records were retained, even if other data was unavailable for certain players.

The merged file was further integrated with the sentiment data by using player name as key column. A left join operation was applied, using the merged dataset as the base table.

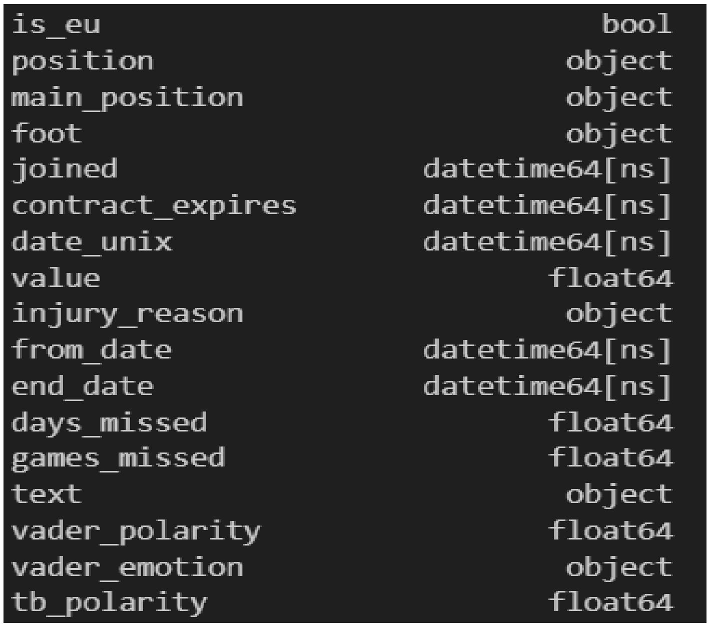
```
# Function to clean player_name
def clean_player_name(name):
    if pd.isna(name):
        return name
    name = str(name)

    name = re.sub(r"\(.*?\)", "", name)   # remove anything inside brackets:
(1), (123), etc.
    name = re.sub(r"\d+", "", name)       # remove any numbers
    name = re.sub(r"^\*", "", name)       # remove * only if it is at the
START
    name = unicodedata.normalize("NFKD", name)  # Remove accents / diacritics
(Ş → S, Ü → U, Ö → O)
    name = name.encode("ascii", "ignore").decode("utf-8")
    name = name.strip()                   # remove extra spaces
    name = name.upper()                   # convert to UPPERCASE

    return name

# Apply to player_name column
df["player_name"] = df["player_name"].apply(clean_player_name)
```

The variation in the player name format was found like player profile have player name as John Thompson (10001). Whereas the sentiment data was having player name in the format like John Thompson. This was leading to invalid matches. Before merging the name was formatted properly.

Data cleaning:

The dataset contained missing values in several columns. Injury-related attributes such as days missed and games missed were filled with zero, assuming no recorded injuries for those players. Other numerical attributes were handled using appropriate statistical methods such as mean, median, or mode depending on their distribution.

Several columns were stored in incorrect formats. Count-based variables such as goals**,** assists**,** cards**,** and minutes played were converted to integer values. Date-related columns including date of birth, joining date, and contract expiry date were converted into datetime format to enable proper time-based analysis.



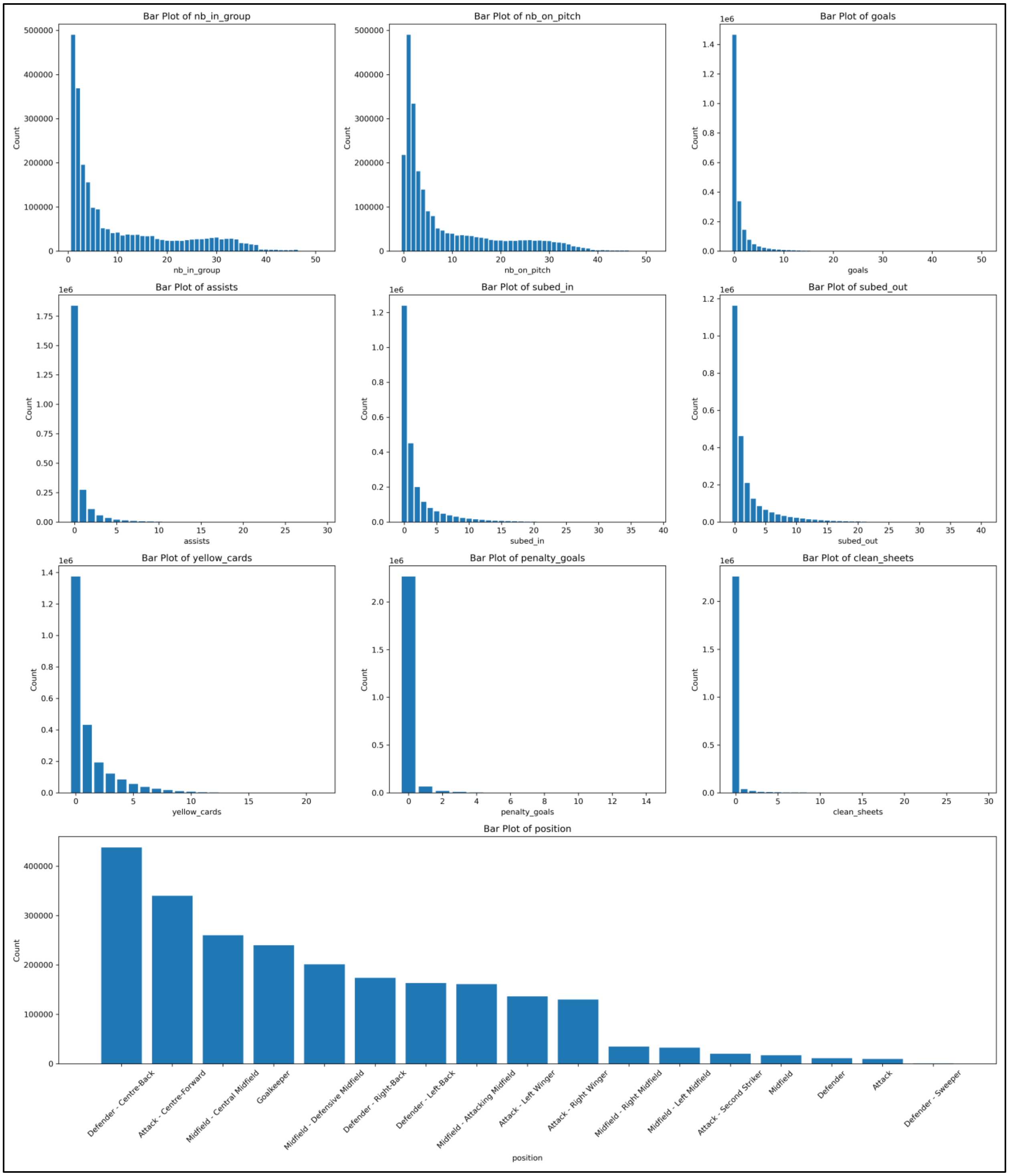
```
is_eu                               bool
position                          object
main_position                     object
foot                              object
joined                    datetime64[ns]
contract_expires          datetime64[ns]
date_unix                 datetime64[ns]
value                            float64
injury_reason                     object
from_date                 datetime64[ns]
end_date                  datetime64[ns]
days_missed                      float64
games_missed                     float64
text                              object
vader_polarity                   float64
vader_emotion                     object
tb_polarity                      float64
```

During data validation, invalid values such as negative numbers were identified in several count-based features (e.g., matches played, goals, assists), which are logically constrained to non-negative integers. These anomalies likely arose due to data entry errors or inconsistencies during data aggregation. Such values were corrected by replacing them with valid non-negative representations to ensure numerical and semantic correctness of the dataset.Additionally, certain attributes that are inherently integer-based were found to contain decimal values.

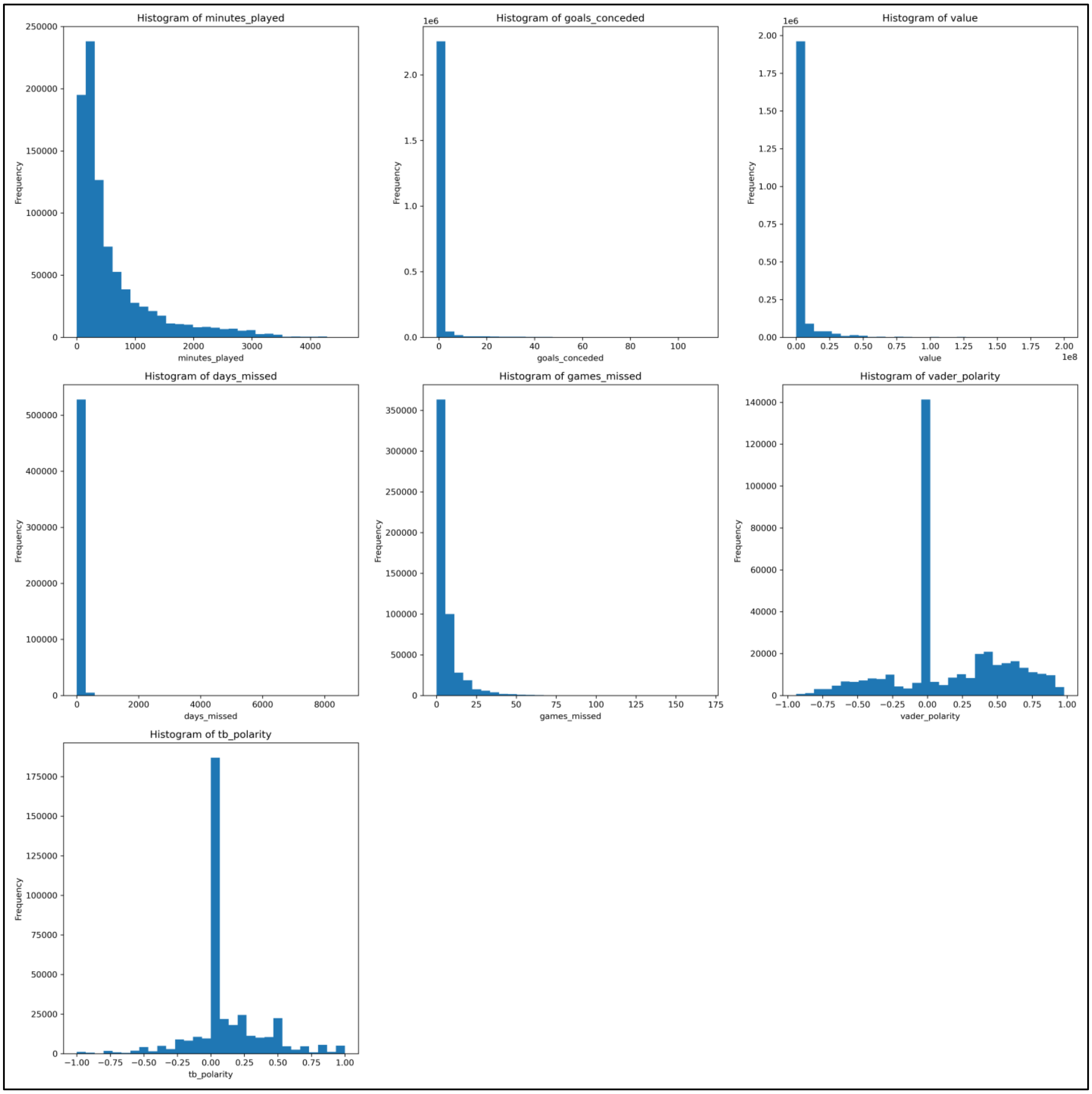
Outlier analysis was conducted on continuous variables such as minutes played, market value, and player height, which exhibited extreme values due to exceptional performances, rare player profiles, or reporting inconsistencies. Rather than removing these observations—which could result in the loss of valuable and representative information—outliers were handled using percentile-based capping (winsorization). Values beyond predefined upper and lower percentile thresholds were capped to those limits, thereby reducing the influence of extreme values on model training while retaining the overall data distribution and important variability.

Sentiment polarity scores generated using the VADER and TextBlob (TB) sentiment analysis techniques were carefully validated to ensure they lay within their respective valid ranges. This verification step helped identify any anomalous, missing, or out-of-range values that could arise due to preprocessing errors, incomplete text data, or sentiment extraction failures. Such invalid or missing sentiment scores were appropriately handled during preprocessing through correction, imputation, or exclusion, ensuring the reliability and consistency of sentiment-related features.

s



To further assess data quality, various graphical techniques were employed to identify outliers, anomalies, and inconsistencies across numerical features. These visualizations provided intuitive insights into data distributions, highlighted extreme values, and supported informed decisions regarding outlier handling and data normalization.



After completing the cleaning process, the dataset was re-checked to confirm the absence of missing values, invalid entries, and inconsistencies. The cleaned dataset was then considered suitable for exploratory data analysis and machine learning tasks.

A memory-efficient data integration pipeline was implemented using chunk-based processing to merge player profiles, market values, injury history, and performance data. Datatypes were optimized to reduce memory usage, latest market values were retained to avoid data leakage, and chunk-wise merging ensured scalability for large datasets.

Data type Conversion:

Several attributes required conversion to appropriate data types. Date-related columns (such as date of birth, contract start, and contract expiry) were converted to datetime format. Derived features such as player age were calculated from date fields to provide more meaningful numerical inputs for the model.

Feature Transformation:

To reduce skewness and improve model stability, transformations were applied to selected numerical features:

Logarithmic and power transformations were applied to highly skewed variables such as market value in order to reduce skewness and stabilize variance. These transformations helped make the data distribution more symmetrical, improving the model's ability to learn meaningful patterns and reducing the influence of extreme values.

Additionally, scaling techniques such as standardization were employed to bring all numerical features onto a comparable scale. This ensured that features with larger magnitudes did not disproportionately influence the learning process, thereby enhancing model stability, convergence, and overall predictive performance.

Encoding Categorical Variables:

Categorical variables were transformed into numerical representations to make them suitable for machine learning algorithms.

Categorical variables were encoded using appropriate techniques based on their nature. Binary categorical features were transformed using label encoding, as this approach efficiently represents two-category variables without introducing ambiguity.

For multi-class categorical variables, one-hot encoding was applied to create separate indicator variables for each category, thereby preventing the introduction of unintended ordinal relationships and ensuring that the encoded features were interpreted correctly by machine learning models.

Feature Selection and Reduction:

Irrelevant and redundant features that did not contribute significantly to prediction were removed to reduce model complexity and improve generalization.

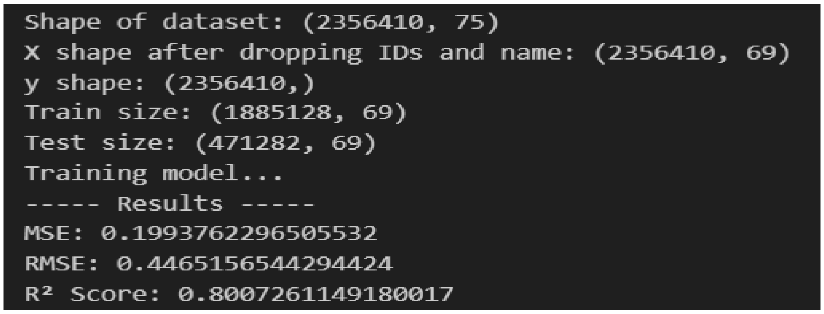
Lasso regression was used for feature selection. Along with it feature importance scores derived from tree-based models were analyzed to quantify the relative contribution of each feature toward the prediction.

Model Development and Experimentation:

To identify the most suitable model for predicting player transfer value, multiple machine learning algorithms were implemented and evaluated. Each model was selected based on its ability to capture different types of relationships within the data. The models were trained using the same preprocessed dataset to ensure a fair comparison.

1. Linear Regression

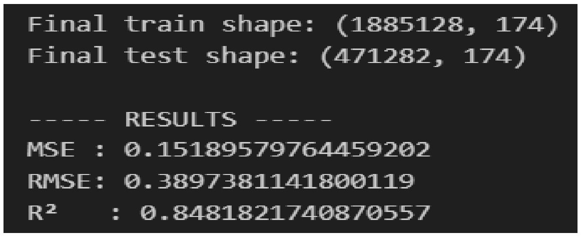
Linear Regression was used as a baseline model to understand the linear relationship between the input features and the target variable. This model assumes a direct proportional relationship between predictors and the output. Although simple and interpretable, Linear Regression is limited in handling complex, non-linear patterns present in real-world data. The performance of this model provided a reference point for evaluating more advanced techniques.



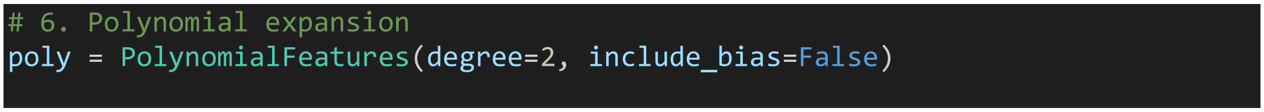
```
Shape of dataset: (2356410, 75)
X shape after dropping IDs and name: (2356410, 69)
y shape: (2356410,)
Train size: (1885128, 69)
Test size: (471282, 69)
Training model...
----- Results -----
MSE: 0.1993762296505532
RMSE: 0.4465156544294424
R² Score: 0.8007261149180017
```

2. Polynomial Featurisation

To capture non-linear relationships, polynomial feature transformation was applied to the input variables, followed by Linear Regression. By introducing polynomial terms, the model was able to learn curved relationships between features and the target variable. While this approach improved performance compared to basic Linear Regression, higher-degree polynomials increased the risk of overfitting, making careful selection of polynomial degree essential.



```
Final train shape: (1885128, 174)
Final test shape: (471282, 174)

----- RESULTS -----
MSE : 0.15189579764459202
RMSE: 0.3897381141800119
R²   : 0.8481821740870557
```

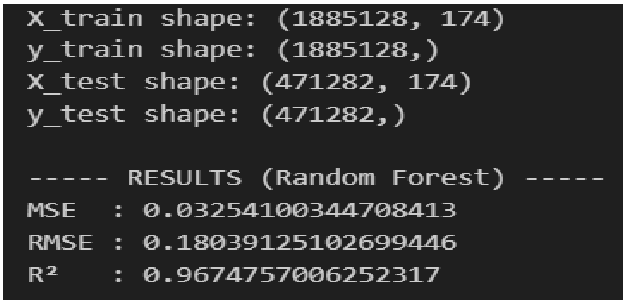


```
# 6. Polynomial expansion
poly = PolynomialFeatures(degree=2, include_bias=False)
```

```
X1_train_poly = poly.fit_transform(X1_train_scaled)
X1_test_poly = poly.transform(X1_test_scaled)
```

3. Random Forest Regression

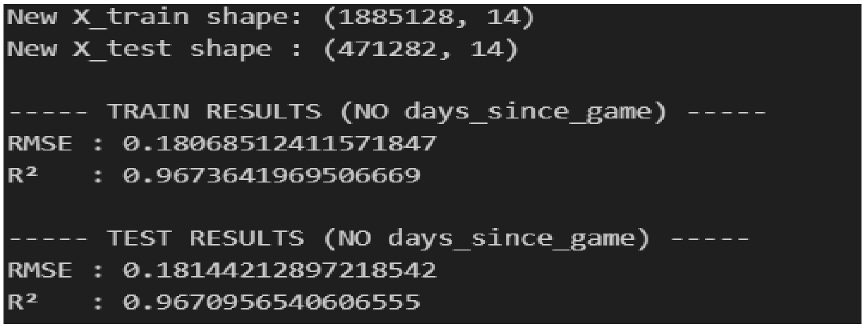
Random Forest Regression, an ensemble learning method, was implemented to improve prediction accuracy and robustness. This model builds multiple decision trees using random subsets of data and features, and the final prediction is obtained by averaging the results. Random Forest effectively handles non-linearity, feature interactions, and outliers, making it well-suited for complex datasets. Additionally, it provides feature importance scores, helping in understanding the influence of individual features on player transfer value.



```
X_train shape: (1885128, 174)
y_train shape: (1885128,)
X_test shape: (471282, 174)
y_test shape: (471282,)

----- RESULTS (Random Forest) -----
MSE  : 0.03254100344708413
RMSE : 0.18039125102699446
R²   : 0.9674757006252317
```

```
# Get feature importances from the trained model
importances = rf_model.feature_importances_

# Convert to a pandas Series with feature names
feature_importance_series = pd.Series(importances, index=X_train.columns)

# Sort features by importance (descending)
feature_importance_sorted =
feature_importance_series.sort_values(ascending=False)

# Show top 20 important features
top_n = 20
print(f"\n----- TOP {top_n} FEATURES -----")
print(feature_importance_sorted.head(top_n))

# Save all feature importances
feature_importance_sorted.to_csv("feature_importances.csv", header=True)
```

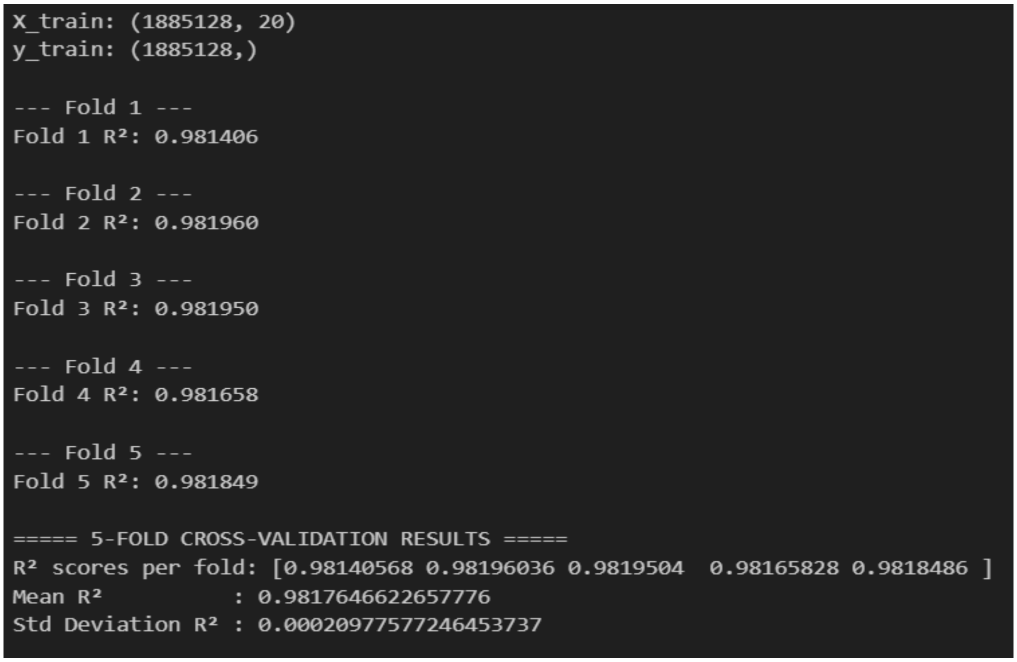
Trying with some selected features:



```
New X_train shape: (1885128, 14)
New X_test shape : (471282, 14)

----- TRAIN RESULTS (NO days_since_game) -----
RMSE : 0.18068512411571847
R²   : 0.9673641969506669

----- TEST RESULTS (NO days_since_game) -----
RMSE : 0.18144212897218542
R²   : 0.9670956540606555
```

4. XGBoost Regression

XGBoost was employed as an advanced boosting algorithm to further enhance prediction performance. Unlike Random Forest, XGBoost builds trees sequentially, where each new tree corrects the errors of the previous ones. It incorporates regularization techniques to prevent overfitting and efficiently handles large datasets with high predictive power. Due to its ability to model complex relationships and optimize loss functions effectively, XGBoost achieved superior performance compared to other models.

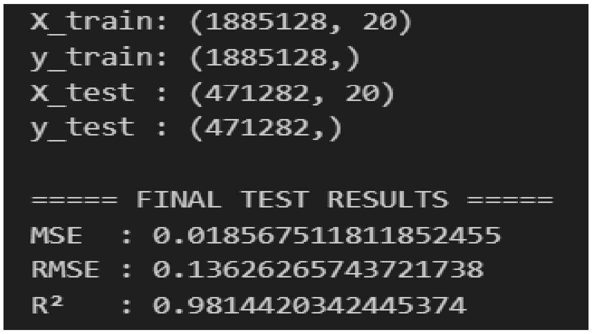


```
X_train: (1885128, 20)
y_train: (1885128,)

--- Fold 1 ---
Fold 1 R²: 0.981406

--- Fold 2 ---
Fold 2 R²: 0.981960

--- Fold 3 ---
Fold 3 R²: 0.981950

--- Fold 4 ---
Fold 4 R²: 0.981658

--- Fold 5 ---
Fold 5 R²: 0.981849

===== 5-FOLD CROSS-VALIDATION RESULTS =====
R² scores per fold: [0.98140568 0.98196036 0.9819504  0.98165828 0.9818486 ]
Mean R²          : 0.9817646622657776
Std Deviation R² : 0.00020977577246453737
```

5. Model Evaluation and Comparison

All models were evaluated using standard regression performance metrics such as:

- R² Score
- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)



```
X_train: (1885128, 20)
y_train: (1885128,)
X_test : (471282, 20)
y_test : (471282,)

===== FINAL TEST RESULTS =====
MSE  : 0.018567511811852455
RMSE : 0.13626265743721738
R²   : 0.9814420342445374
```

The evaluation results demonstrated that ensemble-based models (Random Forest and XGBoost) significantly outperformed linear and polynomial models, indicating the presence of complex, non-linear relationships in the data. Among all the tested models, XGBoost provided the best balance between accuracy and generalization.

Model Deployment and User Interface Development

To make the trained machine learning model accessible and usable in real-world scenarios, the system was deployed using a backend API and an interactive frontend application. The deployment architecture separates model inference logic from user interaction, ensuring scalability, maintainability, and ease of use.

1. Backend Deployment Using FastAPI

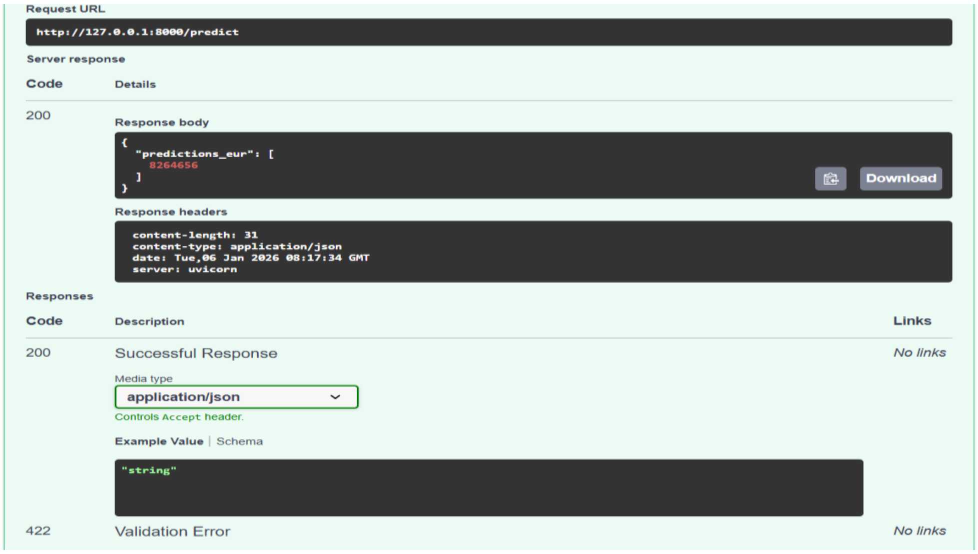
FastAPI was used to deploy the trained model as a RESTful web service. The trained model and preprocessing pipeline were serialized and loaded within the FastAPI application. The API exposes endpoints that accept player-related input features in JSON format and returns the predicted transfer value as the response.

FastAPI was chosen due to its high performance, asynchronous request handling, and automatic generation of interactive API documentation. Input validation was implemented using data schemas to ensure that only valid and correctly formatted data is processed. This backend service allows seamless integration with various client applications and supports real-time prediction requests.

2. API Communication and Inference

When a request is received, the FastAPI server performs the following steps:

1. Validates the input data.
2. Applies the same preprocessing steps used during model training.
3. Passes the processed data to the trained machine learning model.
4. Returns the predicted output in a structured JSON format.
5. This approach ensures consistency between training and inference stages and minimizes prediction errors caused by data mismatch.



Request URL

```
http://127.0.0.1:8000/predict
```

Server response

Code Details

200

Response body

```
{
  "predictions_eur": [
    8264656
  ]
}
```

Response headers

```
 content-length: 31
 content-type: application/json
 date: Tue,06 Jan 2026 08:17:34 GMT
 server: uvicorn
```

Responses

Code	Description	Links
200	Successful Response Media type application/json Controls Accept header. Example Value | Schema "string"	*No links*
422	Validation Error	*No links*

3. Frontend Development Using Streamlit

Streamlit was used to develop a user-friendly web interface for interacting with the deployed model. The frontend application provides input fields such as player statistics, performance metrics, and personal attributes, allowing users to enter data easily without technical expertise.

Streamlit was selected due to its simplicity, rapid development capabilities, and seamless integration with Python-based machine learning workflows. The interface dynamically captures user input, sends it to the FastAPI backend, and displays the predicted transfer value in a clear and visually appealing format.

```python
output = st.empty()  # placeholder for prediction output

if st.button("Predict Value"):
    payload = {"data": [{
        "goals_per_match": goals_per_match,
        "shots_per_match": shots_per_match,
        "injury_count": injury_count,
        "total_days_missed": total_days_missed,
        "average_sentiment": average_sentiment
    }]}
    try:
        response = requests.post("http://127.0.0.1:8000/predict",
json=payload)
        if response.status_code == 200:
            prediction = response.json()["predictions_eur"][0]
            st.success(f"Predicted Player Market Value: €{prediction:,.2f}")
        else:
            st.error(f"Error: {response.status_code} - {response.text}")
    except Exception as e:
        st.error(f"Request failed: {e}")
```

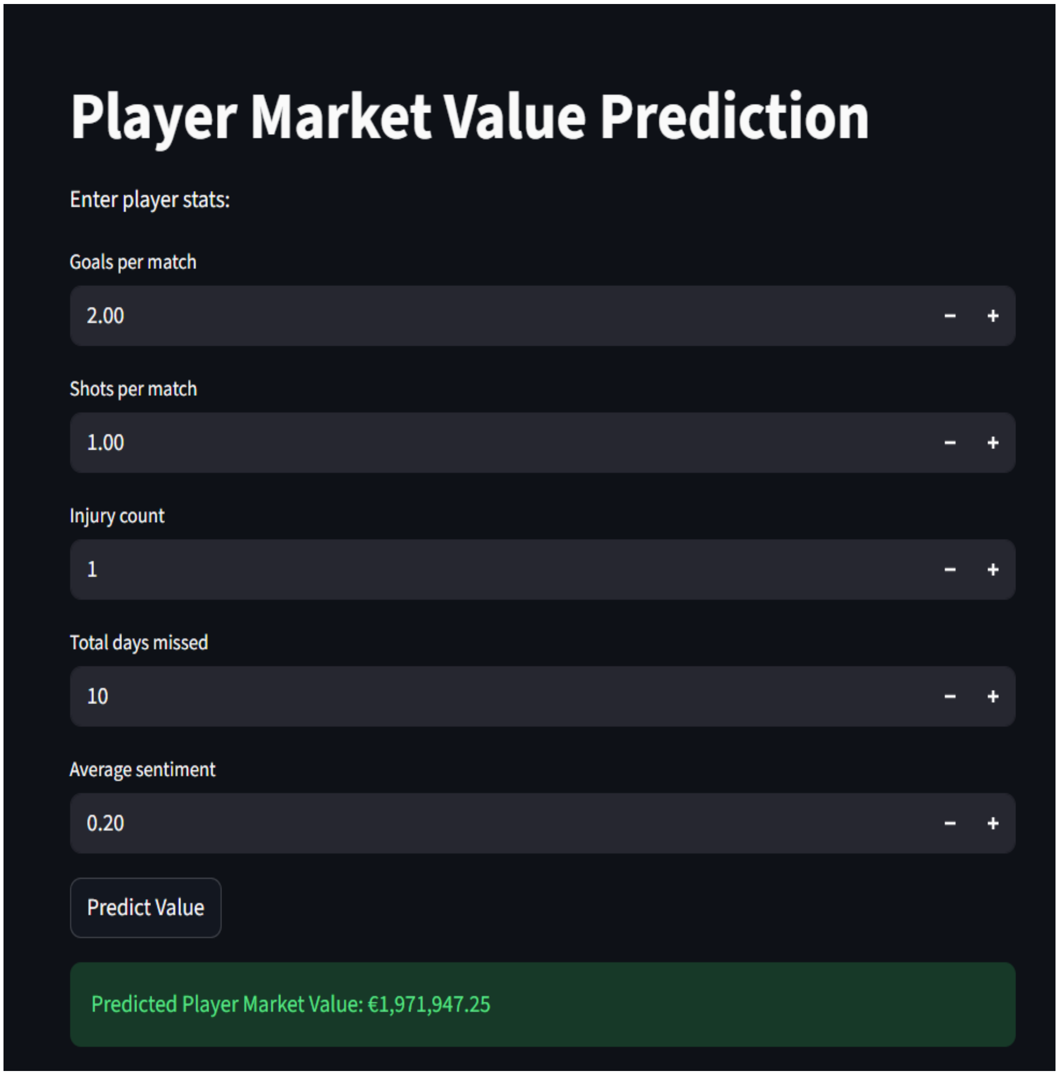
4. Integration Between FastAPI and Streamlit

The Streamlit frontend communicates with the FastAPI backend through HTTP requests. Upon user submission, the frontend sends the input data to the FastAPI prediction endpoint and receives the model output. The result is then rendered on the interface in real time.

This modular architecture ensures a clear separation of concerns:

- **FastAPI** handles model inference and business logic.
- **Streamlit** manages user interaction and visualization.

Such separation improves system scalability and allows independent updates to the frontend or backend without affecting the overall application.



Player Market Value Prediction

Enter player stats:

Goals per match: 2.00

Shots per match: 1.00

Injury count: 1

Total days missed: 10

Average sentiment: 0.20

Predicted Player Market Value: €1,971,947.25