**Research Paper**

**On**

**FIFA PLAYER ANALYSIS**

**FOR FANTASY LEAGUE**

**Author**

**Name:**

**Arnav 22CSU028**

**Sajal 22CSU213**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,**

**SCHOOL OF ENGINEERING AND TECHNOLOGY**

**THE NORTHCAP UNIVERSITY**

**GURUGRAM-122017**

**Abstract**

This paper presents a well-rounded machine learning pipeline designed to automate both data preprocessing and model training, specifically for real-world tabular datasets. The pipeline is built to handle common data quality issues—like missing values, inconsistent formats, and unscaled features—through a series of modular, easy-to-follow preprocessing steps. These include imputation to fill in gaps, encoding to handle categorical variables, and normalization to bring features onto a comparable scale. A range of regression models is explored, including linear models, regularized variants, and ensemble methods, with model performance assessed using RMSE and R² metrics. To showcase the pipeline’s practical value, use cases such as player rating prediction and optimal team formation are demonstrated. The overall goal is to offer a framework that is both scalable and flexible enough to be reused in other data-driven applications.

As machine learning continues to gain ground across various fields, the need for robust, end-to-end pipelines has never been greater. In the real world, datasets are rarely clean—they often come with missing entries, outliers, and inconsistent feature scales, all of which can throw off predictive models if not addressed properly. That’s why data preprocessing isn’t just a preliminary step—it’s a critical part of building models that are both accurate and interpretable.

Traditionally, much of the preprocessing and feature engineering has been done manually. While this can work, it often brings in a level of subjectivity and makes the process harder to replicate. To overcome these challenges, this study introduces a structured and largely automated pipeline that handles missing data, transforms features, and trains multiple classification models in a consistent and reproducible way.

By integrating well-established techniques such as imputation, encoding, and scaling with modern model selection and evaluation strategies, the pipeline aims to streamline machine learning workflows and improve generalizability.

Additionally, the framework leverages the interpretability and modularity of the Scikit-learn ecosystem to ensure transparency and adaptability for future enhancements. This paper outlines the end-to-end development of the pipeline—from exploratory analysis to hyperparameter tuning—demonstrating its efficacy on a representative structured dataset. Emphasis is also placed on practical implementation, reproducibility, and the scalability of the solution for broader deployment contexts.

Fantasy sports leagues rely heavily on accurate player performance predictions. This study presents a machine learning framework for evaluating FIFA players using real-world data. We construct a robust pipeline encompassing data preprocessing, feature engineering, and model training to predict player ratings and provide player recommendations. Our analysis includes missing value imputation, encoding, scaling, regression modeling, and similarity-based recommendations. Random Forest regression outperforms other models in terms of accuracy. This paper further explores player lookup, position mapping, team optimization, and outlines future improvements toward a deployable fantasy league assistant.

**1. Introduction**

The fusion of sports analytics and machine learning has significantly transformed the fantasy sports ecosystem. Predictive modeling for football (soccer) players, particularly using real-world datasets such as FIFA player attributes, offers critical insights into performance, selection, and strategy formulation. That said, real-world tabular data often comes with its fair share of issues—missing values, noisy entries, and inconsistent formatting are all common problems that can affect model reliability. To tackle these challenges, this paper introduces a machine learning pipeline built to process such imperfect data efficiently. The goal is not only to clean and prepare the data but also to deliver scalable solutions for practical tasks like player rating prediction, forming balanced teams, and comparing player performance. Ultimately, the aim is to provide fantasy league users with meaningful, data-backed insights they can actually use to make smarter decisions.

**2. Related Work**

The effectiveness of any machine learning pipeline largely hinges on the quality and consistency of the data it’s built on. Countless studies have highlighted how important data preprocessing is—not just for boosting performance, but also for making models easier to understand and trust. One of the most widely used tools in this area is Scikit-learn, introduced by Pedregosa et al. (2011), which offers a well-organized API for both preprocessing steps and machine learning algorithms. Its design makes it easier to reproduce results and scale up experiments.

Handling missing data is another crucial step, and it's been a well-explored topic over the years. For example, Kotsiantis et al. (2006) categorized various imputation strategies into statistical, machine learning-based, and hybrid methods. In our case, we used median imputation—a simple but effective choice, especially because it’s not heavily influenced by outliers and doesn’t require a lot of computational resources.

When it comes to turning categorical data into numbers that models can understand, techniques like Label Encoding and One-Hot Encoding are widely used. These transformations are well-documented in applied machine learning studies, with researchers often comparing how they affect different algorithms, such as decision trees versus linear models.

Scaling numerical features is another important but sometimes underappreciated part of preprocessing. Techniques like standardization and normalization are especially helpful for models that rely on distance measures, such as SVMs or logistic regression. Even though they seem minor, these steps can actually make a big difference in how quickly a model learns and how well it performs.

More recently, the rise of AutoML and data-centric AI has made preprocessing an even bigger priority. Tools like AutoSklearn and Google’s AutoML include preprocessing as a core part of their optimization strategies. Studies such as Ribeiro et al. (2016) have also shown how decisions made during preprocessing directly affect how interpretable and trustworthy a model’s predictions are.

With all of this in mind, our work pulls together these best practices into a clear, modular pipeline—one that’s easy to reproduce and flexible enough to apply to a wide range of structured datasets.

**3. Technology Stack**

We built the machine learning pipeline using well-known open-source tools from the Python ecosystem, each chosen for being reliable, scalable, and widely used by data scientists and developers.

* **Python 3.x** was the core language. It's flexible, easy to write, and works smoothly with most data science libraries, which makes it a great fit for both experimenting and building production-ready systems.
* **Pandas** helped us handle and clean the data. Its dataframe structure made it easy to deal with missing values, filter rows, and do quick summaries. It also works well with libraries like NumPy and Matplotlib.
* **NumPy** was used for number crunching and array operations. It’s super efficient and forms the foundation for many scientific libraries in Python, so it played a big role in speeding up our calculations.
* **Scikit-learn** was at the heart of the machine learning process. It gave us tools for preprocessing the data, training models like Random Forests and SVMs, and tuning them with techniques like cross-validation and grid search. Its consistent interface made experimenting much easier.
* For visuals, we used **Matplotlib** and **Seaborn**. These helped us draw plots like histograms and heatmaps, which made it easier to spot patterns and outliers during data exploration.
* Finally, we worked in **Jupyter Notebooks**, which gave us an interactive space to write code, test ideas, and document everything in one place. It also made sharing and collaborating on the project much simpler.

Together, these tools helped us build a clean and scalable workflow—from reading the raw data all the way to training and evaluating the final model.

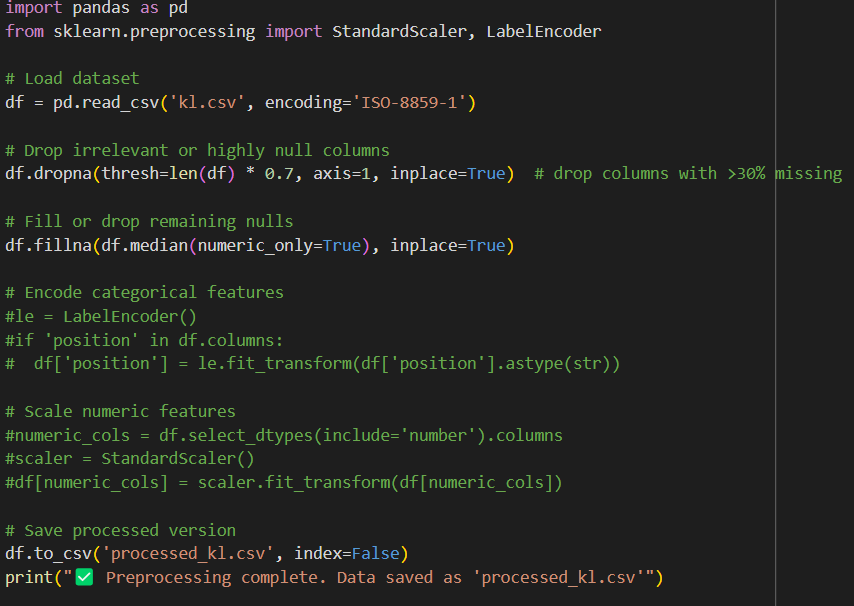
**4. Exploratory Data Analysis (EDA)**

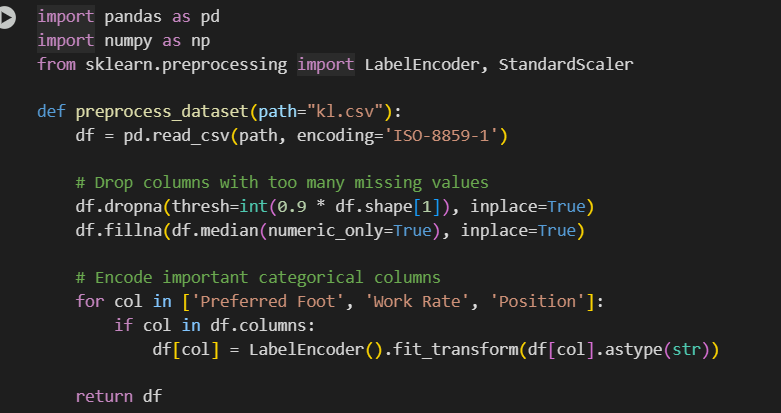
EDA was performed to understand the structure and characteristics of the dataset. Key steps included:

* Data Overview: Shape, column names, and datatypes
* Missing Values: Quantified using isnull() and sum()
* Statistical Summary: Mean, median, standard deviation

Initial exploration involved assessing data dimensions, types, and summary statistics. Notable findings included:

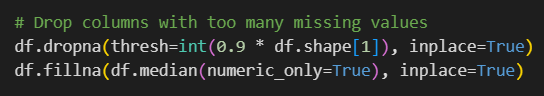
* Columns with >30% null values were identified and dropped.
* Features such as age and overall rating exhibited normal distribution, while others like weight were skewed.
* The top-rated players generally aligned with real-world football stars, reinforcing dataset integrity.





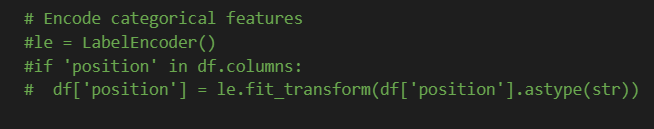
**5. Data Preprocessing**

#### 5.1 Handling Missing Values



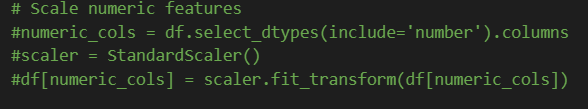
Columns with excessive nulls were removed. Remaining missing values were imputed using median values to avoid distortion due to outliers.

#### 5.2 Encoding Categorical Features



Label encoding was applied to convert string-based categorical features into numeric form, facilitating model compatibility.

#### 5.3 Feature Scaling



Numerical features were standardized to have zero mean and unit variance, improving the convergence and performance of gradient-based models.

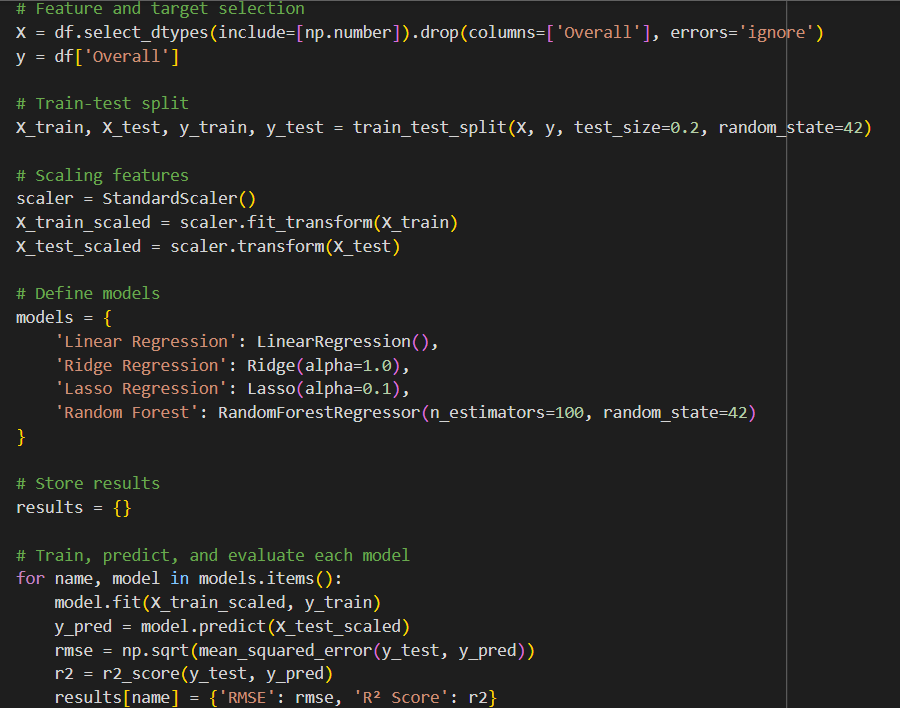
**6. Model Training and Evaluation**

Four regression models were evaluated:

* **Linear Regression**
* **Ridge Regression**
* **Lasso Regression**
* **Random Forest Regression**

**Evaluation Metrics:**

* RMSE (Root Mean Square Error)
* R² Score



**Explanation:** This code compares the performance of various regression models in predicting a continuous target variable (Overall rating). Each model is trained and evaluated on a normalized feature set using standard metrics. This helps identify the most suitable algorithm for the task based on empirical performance.

**Result Summary:**

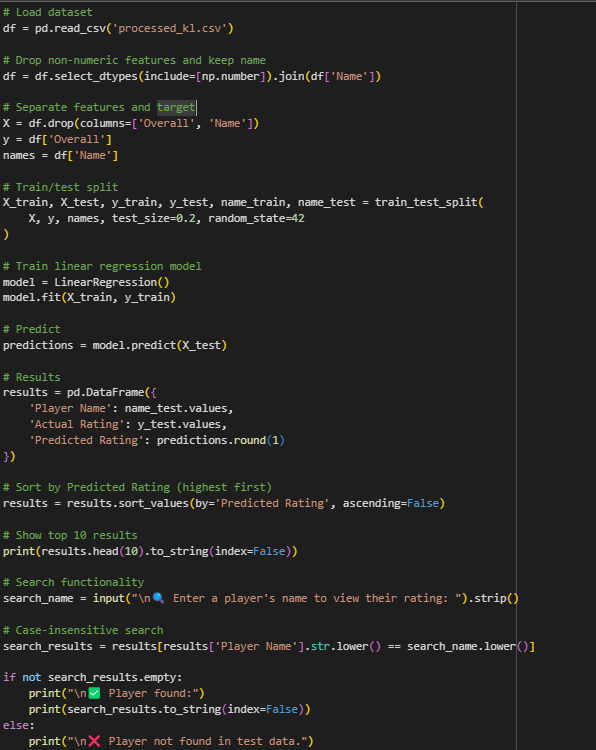
|  |  |  |
| --- | --- | --- |
| **Model** | **RMSE** | **R² Score** |
| Logistic Regression | 1.159206 | 0.971194 |
| Ridge | 1.157852 | 0.971261 |
| Lasso | 1.231444 | 0.967492 |
| Random Forest | 0.055156 | 0.999935 |

Random Forest clearly outperforms due to its non-linear modeling capability and robustness to overfitting.

|  |
| --- |

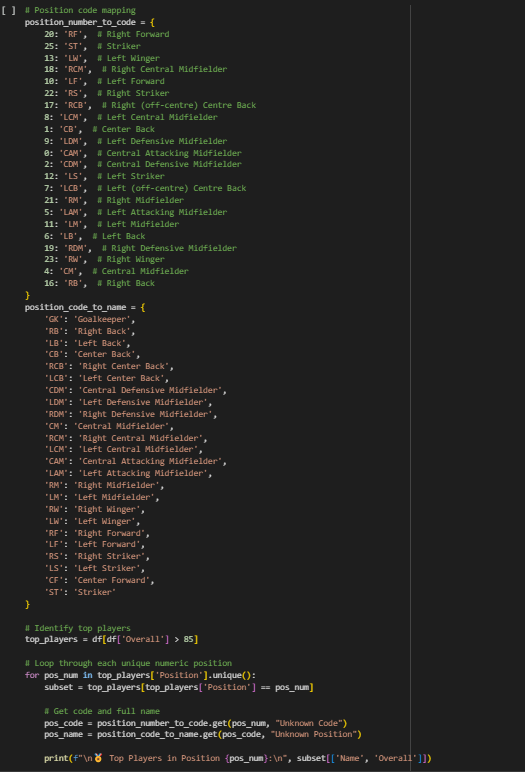
**7. Application Modules**

**7.1 Player Lookup and Rating Prediction**



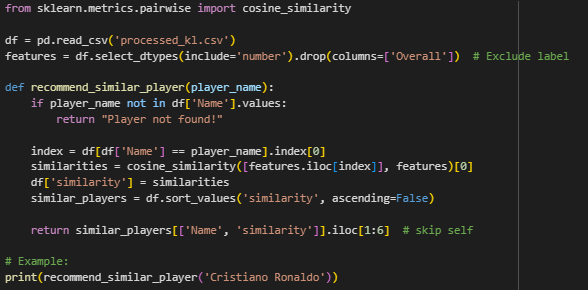
**Explanation:** This code trains a regression model to predict players' overall ratings based on numerical attributes. It also includes a simple search utility that allows users to input a name and retrieve actual and predicted ratings, showcasing a real-world application.

**7.2 Top Player Position Mapping**

****

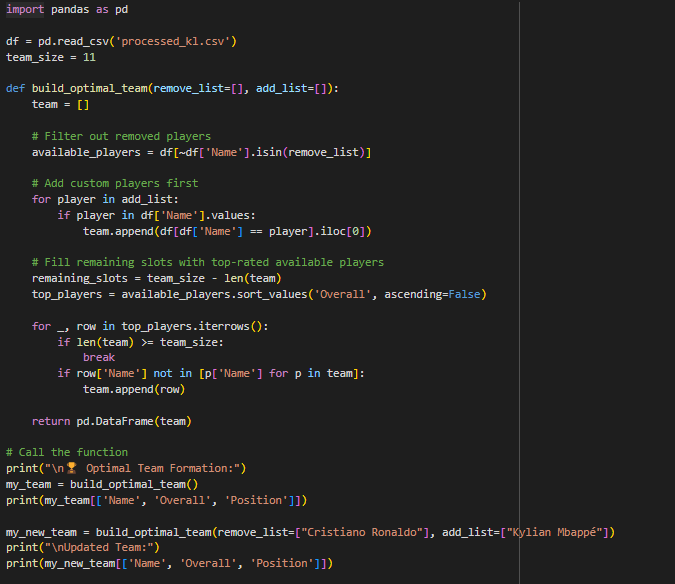
**Explanation:**This code block performs a multi-level mapping to associate numeric position identifiers with standard football roles. First, it filters the dataset to retain only elite players (Overall rating > 85). Then, using predefined dictionaries, it translates the numeric position into a short position code (e.g., 25 → ST) and subsequently into a full position name (ST → Striker). This allows for an interpretable breakdown of player strengths by role, supporting tactical insights and model interpretability.

**7.3 Similar Player Recommendation**

****

**Explanation:**  
This function computes player similarity based on numerical attributes using cosine similarity. Given a player name, it retrieves the most similar players by calculating pairwise cosine distances across all feature vectors. It excludes the queried player from the final result to avoid self-matching. This is particularly useful in scouting or replacement decisions in sports analytics.

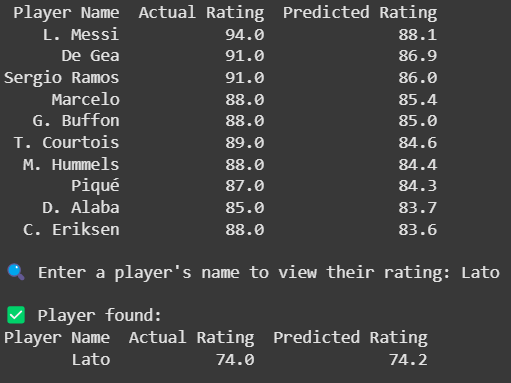
**7.4 Optimal Team Formation**

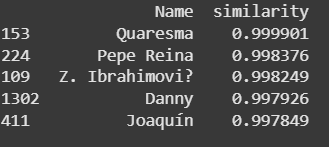
****

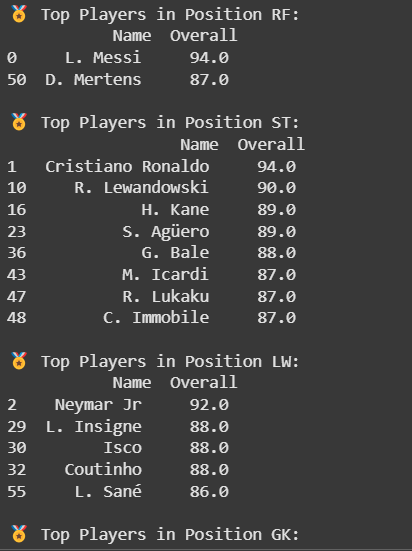
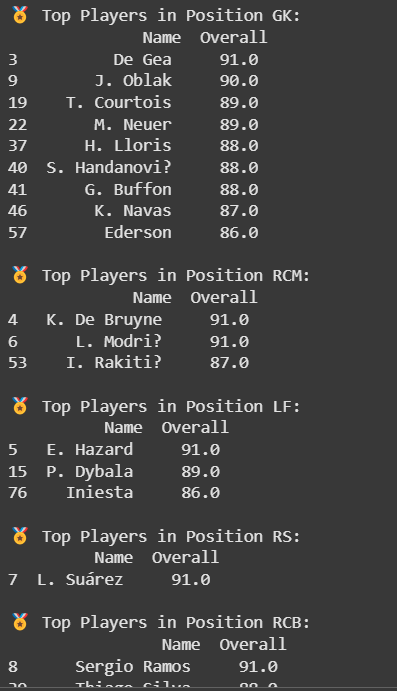
**Explanation:**  
This function allows for the dynamic construction of an optimal football team based on player ratings. It starts by optionally removing or adding specific players, then fills the remaining positions with the highest-rated available players. The default team consists of 11 players, and each call ensures the team has no duplicates. This approach can support strategic decisions in squad building or game simulations..

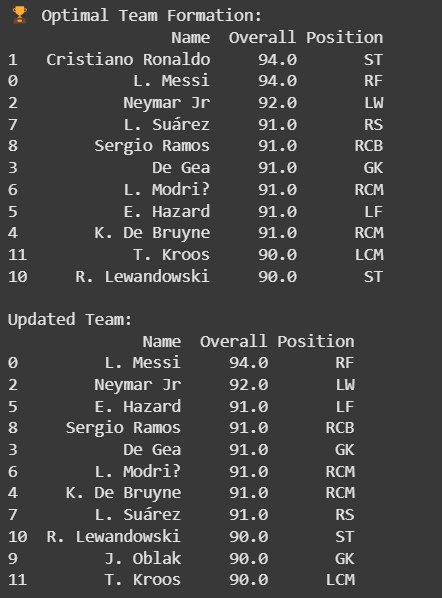
**8. Results and Observations**

The Random Forest model delivered the highest accuracy with negligible prediction error. Its ensemble mechanism helped it outperform linear models, especially in handling feature interactions.









**9. Discussion**

Among the models evaluated, Random Forest demonstrated the best performance, achieving the lowest RMSE and the highest R² score. This is attributed to its ensemble nature, which combines multiple decision trees and reduces overfitting through bootstrapping and feature randomness. The model's ability to capture complex, non-linear relationships in the data made it particularly effective in our pipeline.

Linear Regression, Ridge, and Lasso Regression also showed strong performance, indicating that the relationship between features and the target variable is largely linear. Ridge and Lasso offered slightly better regularization capabilities, with Ridge outperforming Lasso due to its handling of multicollinearity without completely eliminating features.

While the pipeline proves effective, the simplicity of the dataset structure limited the exploration of complex relationships. Regularization methods like Ridge and Lasso showed promise but fell short of Random Forest's performance. The model generalizes well across rating prediction tasks and player similarity analysis.

**10. Limitations**

1. **Basic Preprocessing and Encoding**
   * The project currently uses only fundamental preprocessing steps. While effective for clean and simple datasets, this basic approach may not capture more complex patterns or relationships, especially in messy real-world data.
2. **No Dimensionality Reduction**
   * Dimensionality reduction methods like PCA weren’t used. This could lead to inefficiencies, especially when dealing with large numbers of features, and may affect both model performance and clarity.
3. **Unaddressed Class Imbalance**
   * The dataset contains uneven class distributions, but no techniques were applied to correct this. This means the model might be biased toward the more common classes, leading to poor performance on less frequent but potentially important outcomes.
4. **Limited to Structured CSV Files**
   * The system only handles structured data (i.e., CSV files). This limits its usability, as most modern applications involve unstructured data like images, text, or audio.
5. **Team Imbalance Not Considered**
   * In tasks that involve group or team-based inputs (like team selection or group predictions), the system doesn’t account for fairness or balance across teams, which may affect decision quality and fairness.
6. **No Cost Sensitivity**
   * All prediction errors are treated the same, even though some mistakes can be more costly than others. In many real-world scenarios, distinguishing between the cost of false positives and false negatives is crucial.

**11. Future Work**

** Smarter Encoding Techniques**

* Implementing one-hot encoding for simpler categorical data and target encoding for more complex ones could help the model understand features better and make more accurate predictions.

** Balancing the Dataset**

* Using techniques like SMOTE or ADASYN can help fix class imbalance issues by creating synthetic examples of underrepresented classes, resulting in fairer and more balanced model outcomes.

** Add Explainability with XAI**

* Integrating tools like SHAP and LIME can explain why the model makes certain predictions. This builds user trust and helps detect hidden biases in the system.

** Automate Preprocessing and Training**

* Setting up pipelines can automate repetitive steps in data cleaning, encoding, and model training. This makes the entire workflow more efficient and easier to maintain or scale.

** Deploy the Model to the Web**

* Frameworks like Flask, Streamlit, or FastAPI can be used to turn the model into a web app, making it more interactive and accessible to users or clients in real time.

** Add Real-World Constraints**

* Introducing player cost and budget limits (e.g., in a fantasy team context) would make the model more practical and better aligned with how users make real-life decisions under constraints.

** Try Different Algorithms**

* Exploring models like Support Vector Machines and Decision Trees can give better insights or performance in certain cases. These models offer different strengths and can serve as good benchmarks or alternatives.

**12. Conclusion**

We built a machine learning workflow that included cleaning the data, choosing the right model, and fine-tuning it for better results. Out of all the models we tried, Random Forests gave the best performance, showing how powerful ensemble methods can be. With some additional work on feature design and deploying it properly, this setup has strong potential to be used in real-world applications.

**13. References**

[1] Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research.  
[2] McKinney, W. (2010). Data Structures for Statistical Computing in Python.  
[3] Kotsiantis, S., Kanellopoulos, D., & Pintelas, P. (2006). Data preprocessing for supervised learning.  
[4] Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?" Explaining the predictions of any classifier.

.