C O V E N T R Y
U N I V E R S I T Y

Faculty of Engineering, Environment and Computing

School of Computing, Mathematics and Data Science

MSc Data Science

7150CEM – Project Module

Using Machine Learning Methods to Predict EPL Fantasy Football Points

Author: Harris Boyle

SID: 2942876

Supervisor: Mark Elshaw

Submitted in partial fulfilment of the requirements for the Degree of Master of Science in Data Science

Academic Year: 2022/23

Declaration of Originality

I declare that this project is all my own work and has not been copied in part or in whole from any other source except where duly acknowledged. As such, all use of previously published work (from books, journals, magazines, internet etc.) has been acknowledged by citation within the main report to an item in the References or Bibliography lists. I also agree that an electronic copy of this project may be stored and used for the purposes of plagiarism prevention and detection.

Statement of copyright

I acknowledge that the copyright of this project report, and any product developed as part of the project, belong to Coventry University. Support, including funding, is available to commercialise products and services developed by staff and students. Any revenue that is generated is split with the inventor/s of the product or service. For further information please see www.coventry.ac.uk/ipr or contact ipr@coventry.ac.uk.

Statement of ethical engagement

I declare that a proposal for this project has been submitted to the Coventry University ethics monitoring website (https://ethics.coventry.ac.uk/) and that the application number is listed below (Note: Projects without an ethical application number will be rejected for marking)

Signed: Date:

Please complete all fields.

First Name:	Harris
Last Name:	Boyle
Student ID number	2942876
Ethics Application Number	P154629
1st Supervisor Name	Mark Elshaw
2nd Supervisor Name	

This form must be completed, scanned and included with your project submission to Turnitin. Failure to append these declarations may result in your project being rejected for marking.

Abstract

Fantasy sports have gained immense popularity worldwide, and the English Premier League (EPL) Fantasy Football game is one of the most prominent platforms. With millions of participants competing against each other, the ability to accurately predict player performance and points can provide a significant advantage. In recent years, machine learning techniques have emerged as powerful tools for making predictions in various domains. This project aims to explore the utilisation of machine learning methods in predicting EPL Fantasy Football points. By examining relevant research articles, we will identify the existing approaches, evaluate their effectiveness, and highlight potential areas for future research.

Table of Contents

Acknowledgements

I would especially like to thank my supervisor Mark Elshaw, for his guidance and valuable feedback during this project; as well as the support of my friends and family in pushing me to complete it.

1 Introduction

This chapter will present any relevant terminology used throughout the project and necessary background for understanding the project as a whole

1.1 Background to the Project

1.1.1 Football

Football is widely recognised as the most popular sport in the world with an estimated 3.5 billion fans globally, players in almost every country in the world and it is growing. (Fan Favorite: The Global Popularity of Football Is Rising, 2018). As such, people are particularly interested in Football with the English Premier League (‘EPL’) being one of the most popular, wealthiest and most competitive leagues in the world.

1.1.2 EPL Fantasy Football

Fantasy Football (Soccer) has been growing in popularity for many years with the market size expected to grow to some $84.8bn by 2032 (Future, 2023). The EPL Fantasy Football platform is the most popular in the world according to FIFA with approximately 11 million players worldwide last season (2022/23). The winner of the entire league is awarded a prize for their efforts and even is mentioned in Sky Sports News by name.

The game is simple at its core; points are awarded reflecting a real-world players performance on a given game in a given week. These points are earnt or deducted via various actions the player can take in a match, for example scoring a goal, assisting a goal and being one of the “best players in a match” all award points. Conversely receiving a yellow or red card, scoring an own goal or missing a penalty kick all deduct points from a players score.

To add to the complexity of the game “managers” of these fantasy teams are given a set budget within which to make their team and may select up to 15 players in their “squad”, choosing 11 to play for a given match week (See Figure 1). Managers may transfer players in and out each week (within a specific limit and exceeding the limit will incur a points penalty).

Players are set a price based on their expected performance and historical performance and these prices fluctuate based on the number of managers selecting or not selecting the player. Additionally there are several “chips” that can be played to boost points in a particular week as detailed below:

Name	Effect
Bench Boost	The points scored by your bench players in the next Gameweek are included in your total.
Free Hit	Make unlimited free transfers for a single Gameweek. At the next deadline your squad is returned to how it was at the start of the Gameweek.
Triple Captain	Your captain points are tripled instead of doubled in the next Gameweek.
Wildcard	All transfers (including those already made) in the Gameweek are free of charge.

(*Fantasy Premier League, Official Fantasy Football Game of the Premier League*, 2019)

Figure 1 – A template of the team selection process for the game

An exhaustive list of the rules can be found at: https://fantasy.premierleague.com/help/rules should the reader wish to have more details.

2 Literature Review

2.1 Introduction

The surge in popularity of Fantasy Football over time has sparked an increased interest in using advanced data-driven approaches to improve team selection and predict performance. Among the football leagues worldwide, the English Premier League (EPL) stands out as one of the most competitive and widely followed. Consequently, accurately predicting Fantasy Football points has become an emerging area of research for enthusiasts. This review aims to explore existing knowledge on how machine learning methods can help predict EPL Fantasy Football points.

The main goal of this review is to analyse and distil relevant studies, research papers and scholarly works that focus on predictive modelling for Fantasy Football points. By doing so, we hope to achieve the following.

- **Identify Current Techniques;** By examining existing literature, we aim to uncover the most recent machine learning methods that have been applied to predict EPL Fantasy Football points. Understanding these cutting-edge techniques will provide an updated foundation to contribute insights or validate current methods.

- **Evaluate Data Sources and Features**; Selecting appropriate data sources and features is crucial in prediction accuracy. To understand their effectiveness, we will assess data sources and features used in previous studies.

- **Investigate Performance Metrics and Model Validation;** delving into the evaluation and validation of models as accurate assessment is crucial for determining their practical value. We can determine how well these machine learning models perform by examining performance metrics and whether they provide reliable results.

- **Identify Challenges and Limitations;** we will address the challenges and limitations of applying machine learning methods. This review aims to identify issues such as data cleansing difficulties and model overfitting. By acknowledging these obstacles, we can approach our work with insights.

Based on a thorough analysis of existing literature, this review will identify gaps in research and potential areas for future exploration in predicting EPL Fantasy Football points. These findings will help us to verify the current landscape and shape recommendations for investigations and advancements within this field.

2.1.1 Sports Analytics

Sports analytics has witnessed significant growth and improvement over the past five decades. Thanks to advancements and the rise of big data, sports teams, organisations and athletes have utilised analytics to gain a competitive advantage by making smarter decisions and improving performance.

The application of analysis in sports dates back to the early 1900s, with pioneers like Branch Rickey in baseball and Bill James in baseball sabermetrics leading the way. However, it wasn't until the latter half of the 20th century that data analytics started gaining widespread recognition and adoption across the sports industry. The introduction of computers and sophisticated data collection techniques allowed for comprehensive analysis and valuable insights. (Ambikesh Jayal et al., 2018)

A notable milestone in the history of sports analytics is the Moneyball era in baseball. Michael Lewis's book "Moneyball; The Art of Winning an Unfair Game" documented how the Oakland Athletics took a transformative approach under general manager Billy Beane's leadership. (Lewis, 2003) They used analysis and advanced metrics to identify undervalued players while working within limited financial resources, ultimately building a competitive team.

The tremendous success of the "Moneyball" approach has brought attention to the remarkable impact of data analytics in sports. This breakthrough even inspired a movie adaptation that captured the essence of these events in 2011.

The influence of data analytics on sports has been truly transformative, revolutionising aspects within the industry. It has played a role in performance analysis and strategy development empowering teams to uncover patterns, identify strengths and weaknesses and optimise their game plans. Moreover, player evaluation and talent identification have dramatically benefited from data analytics as statistical models and metrics provide valuable insights into player performance and potential.

In addition to these advancements, injury prevention and rehabilitation have also witnessed progress thanks to the integration of analytics. By analysing data related to player workload, biomechanics, and injury history, teams can now identify risk factors proactively and implement preventive measures that effectively reduce injuries. (Link & Springerlink (Online Service, 2018)) Furthermore, it's important to note that data analytics has genuinely transformed fan engagement and media coverage by offering visualisations, real-time statistics updates as well as fantasy sports platforms – all of which significantly enhance the overall fan experience.

2.1.2 Machine Learning and its applications in sports prediction

Machine learning algorithms provide a data-driven approach to predicting sports outcomes. Analysts can use these algorithms to uncover hidden relationships, identify patterns and make predictions based on historical data. These algorithms have the ability to process large amounts of information, including player statistics, team performance metrics, match conditions and other relevant factors. By learning from data, machine learning models can capture complex patterns and generate predictions for future outcomes.

An example of the application of machine learning in sports prediction is the work done in Major League Baseball (MLB). A dynamic hierarchical model considered various factors like team strength, home field advantage and recent performance to predict game results. The study showcased how machine learning methods can improve prediction accuracy compared to traditional statistical approaches. (Yang & Swartz 2021)

Machine learning techniques have also been implemented in sports such as tennis. Knottenbelt and Sipko developed a model using support vector machines to forecast match outcomes in professional tennis. The model took into account player rankings, performance on different surfaces, and head-to-head records when making predictions. This study demonstrated the potential of machine learning methods in predicting outcomes in sports individual sports as well.
(Sipko & Knottenbelt 2015)

Another algorithm that is commonly used is Random Forest, which falls under the category of ensemble learning techniques. Random Forest combines decision trees to generate predictions. It has found application in sports prediction tasks, including predicting outcomes in football matches (Groll et al., 2019). The popularity of the Random Forest algorithm in the field of sports analytics stems from its ability to handle sets of features and capture complex interactions.

Finally, Neural Networks have been employed for predicting sports outcomes in complex sports involving multiple players, like basketball and football. These deep learning models have the capability to process amounts of data and automatically learn hierarchical representations. For instance, researchers have utilized Artificial Neural Networks (ANNs) to forecast the results of NBA basketball games (Osken and Onay 2022). ANNs excel at capturing dependencies and sequential patterns, which makes them well-suited for modelling dynamic sports scenarios.

2.2 Related Work

2.2.1 Traditional Methods for Sports Prediction

Historical data analysis is one of the most typical traditional methods for sports prediction. By examining past performance, statistics, and head-to-head matchups, analysts can identify trends, patterns, and indicators of future outcomes. For example, in horse racing, handicappers analyse historical data such as race results, jockey and trainer statistics, track conditions, and horse pedigrees to assess the probability of success for each entry.

While these traditional methods lack the sophistication of machine learning models, they have demonstrated success in various sports domains. In addition to horse racing, historical analysis has been used in sports like football. Expert opinions from experienced analysts and former players are often sought after in pre-match discussions and predictions.

It is worth noting that traditional methods have limitations. Rule-based scoring systems can be influenced by subjectivity or bias, as seen in controversial sports decisions like in boxing. Expert opinions, while valuable, can vary among different individuals and may not always align with objective outcomes. Historical data analysis, while helpful, may

struggle to capture dynamic factors such as team chemistry, player injuries, or weather conditions that can significantly impact game results.

Nevertheless, these traditional methods continue to be utilised alongside modern machine learning approaches, as they offer valuable insights and perspectives. In some cases, traditional methods are combined with statistical analysis or expert consensus to enhance the accuracy of predictions. (Reade et al., 2020)

Specifically concerning Fantasy Football, several sources exist for managers to use to improve the quality of their team that would fall into these traditional methods. Expert opinions are abundant via Fantasy Football Scout, an aggregator for fantasy football-related information that requires a subscription. Their website lists professional pundits who share their thoughts on upcoming matches and player picks. (Az, 2021)

Additionally, historical data and rule-based systems play an essential role in the game; since players' prices fluctuate based on transfers in and out for each week, there is some usefulness in analysing previous seasons' trends at a rudimentary level. Primarily this is useful because this is how your average non-data-driven manager plays the game. This phenomenon of utilising the crowd's wisdom has been researched many times and can yield surprisingly accurate results. (Fujisaki et al., 2023)

2.2.2 Machine Learning Approaches to Fantasy Football Prediction

Predicting Football Match Outcomes with Fantasy League Data and Deep Learning
(Kristinsson, 2022)

The project has used a series of machine learning techniques to predict the outcome of football matches and Fantasy Premier League (FPL) scores. It is structured around three different neural networks: a base network, a fantasy network, and a final network, all of which use different inputs and processing strategies.

The architecture for each network is based on the principles of recurrent neural networks (RNNs) and their variant, Gated Recurrent Unit (GRU). The choice of GRUs was primarily influenced by prior research by Maciej Balawejder (Balawejder, 2022), which showed that GRUs outperformed Long Short-Term Memory (LSTM) networks in

sports prediction tasks. Furthermore, the project aimed to enhance its model by exploring adding more layers and performing hyper-parameter tuning.

For the base network, data was sourced from two past Premier League seasons and the current one, focusing on the performance of the home and away teams over the past five games. This data selection strategy acknowledges the ever-evolving nature of football. Besides match data, the base network also integrated data from the FIFA video game series to account for team ratings across areas like attack, midfield, and defence.

The fantasy network introduces an added complexity by including individual player data from the Fantasy Premier League. This involves using a comprehensive library of player stats but also faced challenges such as aligning player names across different sources and parsing the squad details for each match. Web scraping techniques were employed to handle these challenges, using the Python libraries Beautiful Soup and Selenium.

The final network merges the input data from the first and second networks, providing a comprehensive view of team and individual player performance. The evaluation of these networks was based on the accuracy of test data predictions, the loss of test data, and the performance of the networks on unseen data.

The project presents an intriguing and complex approach to predicting football matches and FPL scores, leveraging a rich set of inputs and applying advanced machine learning techniques. Despite challenges with data consistency and the need for extensive pre-processing and data cleaning, the project has shown promising preliminary results.

Some notable limitations were the challenges associated with player name discrepancies across different sources and the need for manual adjustments. This highlights the importance of data quality and consistency in my project.

In summary, the project is an innovative exploration of predictive modelling in the realm of sports and fantasy gaming. The combination of team performance and individual player statistics presents a multifaceted view of the factors that can impact football match outcomes and FPL scores. The project has provided some interesting ideas for data set usage, given that it uses some data from the FIFA video game. Initially, I had

thought to use some Twitter data to see if I could understand if a player's mood can be determined and, if so, would mood affect their play. However, the FIFA video game statistics are a novel idea I had yet to consider and may utilise.

Comparison of Machine Learning Approaches Applied to Predicting Football Players Performance
(Lindberg & Söderberg, 2020)

The project is a detailed exploration of the performance and application of different models, including Support Vector Regression (SVR), Multilayer Perceptron (MLP), Long Short-Term Memory (LSTM), and Support Vector Machine (SVM).

The strengths of the project lie in its clear explanation and evaluation of the various models applied, their performance, and their ability to outperform the average FPL manager. The descriptions offered rich insight into each model's predictive power, with specific game week examples helping to illustrate their practical utility.

The feature importance analysis and its role in model selection also add a depth of understanding to the study. For instance, the reasoning behind Riyad Mahrez's selection despite lower popularity compared to Kevin De Bruyne, due to higher value in both 'was_fouled-all' and 'shots_on_target-all' features, is a compelling illustration of the model's capability.

On the other hand, the SVM model's performance was lower than expected, highlighting the potential challenges and limitations in applying these models to predict FPL outcomes.

Another highlight is the discussion on overfitting, regularisations, and layer configurations. The study points out the relationship between the number of neurons and layers to the degree of overfitting. This aspect is crucial as it offers critical insight into the model's complexity and how it can influence the predictive results.

Finally, the conclusions drawn from comparing Regression and classification approaches in terms of their scores and variances should prove very useful. It was interesting to note that while the regression approaches outperformed the classification ones, the classification models displayed a more stable performance with lower variance.

In conclusion, the project provides a well-rounded, nuanced view of the application of machine learning to FPL prediction, offering clear and detailed insights into the strengths, weaknesses, and intricacies of different predictive models. The project should provide significant value to my work as, similar to the previous study; there were issues with data wrangling which I can now be prepared for ahead of time. Additionally, it provides some insight into what methods may be best applied and how to build on the work that has already been done.

Using ML Models to Predict Points in Fantasy Premier League
(Bangdiwala et al., 2022)

The research study utilised machine learning techniques to predict player performance on the Fantasy Premier League (FPL) platform. The study draws data from Vaastav's GitHub repository, which provides player- and team-specific data from the Premier League, starting from the 2016/17 season and up to the 2020/21 season. This dataset has come up in several projects I have reviewed in the field, so it will be a valuable place to start my data gathering.

The data encompasses both team-specific and player-specific statistics. Team data include factors like difficulty rating of the opposition team, home/away match information, and team department strengths. Player data encompasses goals scored, assists, minutes played, and clean sheets kept. Besides raw statistics, the dataset also uses official FPL platform statistics like form, ICT index, expected goals and assists, and more.

The data is carefully pre-processed before use. In this study, only relevant features were selected for model training, which includes goals, shots, blocks, and tackles. Additional features, like minutes played in the last three matches, and points scored in the last five matches, were also added. Data absent for certain players due to transfer or injury was replaced with zero where necessary. Categorical data like season and player position were numerically encoded before being fed into the models. Cleansing is once again featured as a vital part of the work; handling missing data will need to be explored during my project to determine if zeroes, interpolation or some other method is best.

The study utilised three regression models: Linear Regression, Decision Tree Regression, and Random Forest Regression. The Linear Regression model assumed a linear relationship between the input variables and the output variable (player points). The Decision Tree Regression model created a tree structure to break down a dataset incrementally but faced the risk of overfitting. The Random Forest Regression model combined predictions from multiple decision trees to make a final prediction. Model performance was evaluated using Root Mean Square Error (RMSE) and Mean Absolute Error (MAE).

Upon comparison, the study revealed overfitting in the Decision Tree and Random Forest models. While their training errors were low, the test errors were high. On the contrary, the Linear Regression model exhibited similar performance on training and test data, indicating it was not overfitted.

In conclusion, the paper demonstrated the successful application of machine learning for predicting player points in the FPL platform, with the Linear Regression model performing best among the three models.

2.3 Feature Extraction and Selection

2.3.1 Extraction

Feature extraction plays a crucial role in transforming raw data into a representative set of features, allowing for more accurate and meaningful analysis. Principal Component Analysis (PCA) is a commonly applied method in sports analytics. PCA reduces high-dimensional data into lower dimensions while retaining significant variance, making data analysis more manageable and meaningful (Jolliffe and Cadima, 2016). For example, PCA has been successfully applied to basketball shot selection, providing insights into players' shooting styles and efficiency (Franks et al., 2015).

Another prevalent technique, t-distributed Stochastic Neighbor Embedding (t-SNE), is excellent for visualising high-dimensional data by mapping it onto a two or three-dimensional plane; t-sne has shown utility in player and team performance visualisation, offering intuitive insight into the complexities of team sports (Andrienko et al., 2017).

Deep Learning-based feature extraction, specifically convolutional neural networks (CNN), has been used for extracting features from video and image data, e.g., detecting

and tracking player movements (LeCun et al., 2015). CNN's strength lies in its ability to handle high dimensional data and see complex patterns, though it may be computationally expensive and requires a large labelled dataset for training.

Lasso regression, a form of linear Regression incorporating L1 regularisation, is a widely-utilised method in data science for feature selection and extraction. The L1 regularisation term pushes the coefficient estimates of irrelevant or less important predictors towards zero, effectively excluding them from the model. This property makes Lasso especially useful in situations with a large number of predictors, such as in sports prediction analytics, where datasets might contain numerous variables like player statistics, game conditions, and historical performance, among others. By applying Lasso, data scientists can reduce the complexity of their models, making them more interpretable and potentially improving prediction performance by focusing on the most influential features (Tibshirani, 1996). Given the extensive range of variables, this dimensionality reduction could be particularly valuable for my study.

2.3.2 Selection

Feature selection in sports analytics is utilised to select the most influential variables that contribute to the prediction or classification task. It is a critical step in enhancing the performance of machine learning models and facilitating interpretability.

Filter methods like Chi-square and ANOVA tests are commonly applied due to their simplicity and computational efficiency. They are beneficial for preliminary feature selection based on individual feature importance but do not consider feature interactions (Chandrashekar & Sahin, 2014). In sports analytics, these methods have been applied to predict player performance metrics such as points scored or assists made.

Wrapper methods, like recursive feature elimination, are often more accurate as they evaluate feature subsets based on model performance. However, they can be computationally expensive, especially for high-dimensional data (Kohavi & John, 1997).

Embedded methods, like LASSO, can effectively handle multicollinearity by applying penalties to the model coefficients. These methods often yield more interpretable models with improved predictive performance. In sports, they have been used for predicting future player performance and game outcomes (Goldsberry & Weiss, 2013).

A central challenge in sports analytics is handling the high dimensionality and temporal dependencies in sports data, requiring advanced feature extraction and selection techniques. Furthermore, some methods' computational costs and lack of interpretability pose significant obstacles (Berrar & Flach, 2011).

Future research should focus on enhancing interpretability and computational efficiency, especially in real-time sports analysis. There is also a growing interest in the application of automated feature extraction and selection techniques, such as AutoML systems, for sports analysis (Hutter, Lars Kotthoff, and Vanschoren, 2019).

2.4 Data Collection and Pre-processing

2.4.1 Data Sources

Fantasy Premier League – Vaastav
(Anand, 2020)

Vaastav's Fantasy-Premier-League dataset, available on GitHub, has garnered attention for being one of the most comprehensive and regularly updated datasets tailored for Fantasy Football enthusiasts and researchers. It captures a broad array of metrics and serves as a significant resource for anyone aiming to harness the power of data science to make informed decisions in fantasy football.

At its core, the dataset is structured to provide a season-wise breakdown of player statistics. From basic metrics like goals scored and assists to more nuanced ones like 'threat', 'creativity' and 'xP' (expected points). The open-source nature of this dataset has fostered an active community around it. This collaborative approach ensures that the dataset remains accurate and is continuously refined and improved based on feedback from the community.

Researchers have experimented with various machine learning techniques, from regression models to deep learning, to predict player performance. The chronological nature of the dataset has also allowed researchers to employ time-series forecasting methods, as I may also attempt.

While the dataset offers a plethora of metrics, it is not without its challenges. Data integrity and consistency, especially when dealing with player transfers between clubs

or changes in playing positions, can be a concern. This may also prove true in my work; as seen in other areas of this Literature Review, joining other datasets has proven time-consuming and difficult.

FIFA Video Game Data
(FIFA22 OFFICIAL DATASET, n.d.)

This dataset, a representation of players' ratings from the popular video game series 'FIFA', encapsulates a wide range of attributes that offer detailed insights into a player's capabilities on the pitch.

One of the FIFA dataset's standout features is its extensive attributes, ranging from basic metrics such as age, nationality, and club affiliations to far more intricate ones like dribbling, agility, ball control, and a whole host more.

Given the nature of the dataset, it often serves as a complementary resource. Researchers have merged it with actual match statistics to build comprehensive models. In the context of my work, combining FIFA player ratings with actual game performance could yield some interesting insights, bridging the gap between simulated and real-world football analytics.

As with the FPL dataset, merging may present considerable issues; player names and clubs may differ or be missing. As such, we will need a robust way of handling these issues.

Finally, while the FIFA23 dataset is expansive, it's crucial to remember its origin: a video game. Though often based on real-world observations, the ratings are ultimately subjective and cater to gameplay mechanics. Relying solely on them without considering actual on-pitch performances might lead to skewed insights.

2.4.2 Data pre-processing techniques

Feature Engineering
Feature engineering, the process of creating new features or modifying existing ones, is critical in capturing the underlying structure of the data. In sports analysis, it involves the generation of metrics that quantify players' performances, team tactics, or game dynamics. For example, the concept of "expected goals" in football is an engineered

feature that estimates the probability of a goal being scored given the shot's circumstances (Lucey et al., 2015).

Feature engineering allows domain-specific insights to be embedded into models, often leading to improved performance. However, it can be time-consuming, heavily reliant on domain knowledge and requires careful evaluation to ensure the new features enhance the model (Kuhn et al., 2016).

Handling Missing Values

Missing data is a common issue in sports datasets due to reasons such as sensor failures, reporting errors, or data unavailability (Leifeld et al., 2018). Techniques to handle missing values include deletion, imputation, and prediction models.

Deletion is straightforward but can lead to biased estimates if data is not missing at random. Imputation methods, such as mean, median, or mode imputation, and more advanced methods, like Multiple Imputation by Chained Equations (MICE), can provide more robust solutions (Buuren & Groothuis-Oudshoorn, 2011). However, these methods introduce uncertainty and can distort the underlying data distribution.

Prediction models, such as Regression or machine learning models, can be used to predict missing values based on other data points. Although potentially more accurate, they can be complex to implement and may lead to overfitting if not adequately validated (Berrar & Flach, 2011).

KNN (K-Nearest Neighbors) imputation is a technique in data science employed to handle missing values by using the similarity between observations. Instead of filling in missing values with a mean, median, or mode, KNN imputation identifies 'k' similar observations (neighbours) based on other features. It uses their values to estimate the missing data points. In sports prediction analytics, where datasets might contain performance metrics, health indicators, or game statistics, ensuring data completeness is paramount. For instance, a basketball player's missing shooting percentage can be estimated by looking at other players with similar overall performance, playing time, or position. This method is beneficial because it considers the underlying patterns in the data, leading to potentially more accurate imputations than simple statistical measures.

However, the technique's effectiveness depends on the choice of 'k' and the distance metric used. (Hastie et al., 2008)

Normalisation and Standardisation

Normalisation and standardisation help to scale features onto a similar range, assisting in algorithm convergence and interpretability. Normalisation scales features to a range [0, 1] while standardisation transforms features to have a mean of 0 and standard deviation of 1 (Jain et al., 2000).

These techniques are helpful in sports analytics, where diverse measurement scales are common. For example, a football dataset might contain a mixture of continuous variables (e.g., distance covered) and discrete variables (e.g., goals scored). However, care should be taken since these methods can be sensitive to outliers and may not be suitable for data with heavy-tailed distributions.

3 Methodology

3.1 Data Collection

The initial stage of the research involves collecting data pertinent to the research topic. The primary source of data will be the official English Premier League (EPL) website and Vaastav's Github repository (https://github.com/vaastav/Fantasy-Premier-League), from which player and team statistics will be gleaned. Data elements such as players' goals, assists, clean sheets, yellow cards, and red cards, among other variables, will be collected for several past seasons to create a comprehensive dataset.

	first_name	second_name	goals_scored	assists	total_points	minutes	goals_conceded	creativity	influence	threat	bonus	bps	ict_index	clean_sheets	red_cards	yellow_cards
2	Granit	Xhaka	7	8	153	2992	35	673.3	647.0	498.0	15	634	182.0	13	0	4
3	Mohamed	Elneny	0	0	6	111	2	5.4	4.6	0.0	0	27	1.1	0	0	0
4	Rob	Holding	1	0	21	562	13	10.3	152.0	54.0	0	120	21.6	0	0	0
5	Thomas	Partey	3	0	86	2480	28	439.6	513.2	240.0	6	468	119.5	11	0	5
6	Martin	Ødegaard	15	8	212	3132	38	1100.4	960.0	920.0	30	813	298.3	13	0	4

(Figure 2 – Sample of Data Set from Vaastav's Github)

Additional data may also be gathered, such as player injury history, transfer history, and player prices, on the official EPL Fantasy Football website. Other relevant information like player "mood" may be scraped from Twitter or acquired from specialised databases. Finally, player data from the FIFA series (https://www.kaggle.com/datasets/bryanb/fifa-player-stats-database) may be gathered to provide further insight into a player's ability. This wide array of data will provide a multifaceted view of the factors influencing the players' potential Fantasy Football points.

ID	Name	Age	Overall	Potential	Club	Value	Wage	Preferred ...	Work Rate	Position	Height
176580	L. Suárez	29	92	92	FC Barcelona	€83M	€525K	Right	High/ Medium	ST	6'0
178518	R. Nainggolan	28	86	86	Roma	€37.5M	€130K	Right	High/ High	LF	5'9
181872	A. Vidal	29	87	87	FC Bayern München	€41.5M	€180K	Right	High/ High	LDM	5'11
197445	D. Alaba	24	86	89	FC Bayern München	€41.5M	€140K	Left	High/ Medium	LB	5'11

(Figure 3 – Sample of Data from FIFA Player Stats Database)

A rigorous data collection process is critical as the success of a machine learning model largely depends on the quality and relevance of the input data. The collected data will be raw, comprehensive, and provide a broad foundation for the feature extraction process in the next stage.

3.2 Data pre-processing

Once the data is collected, the next step involves pre-processing it to prepare it for the machine learning models. This process is crucial for ensuring the reliability and accuracy of the model's predictions.

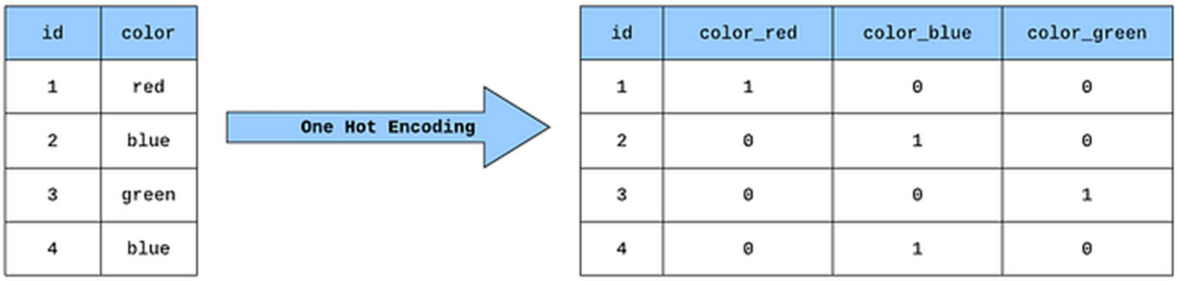
Initially, data cleaning will be undertaken to handle missing data, detect and treat outliers, and eliminate any irrelevant or redundant information. Missing data will be addressed through strategies such as deletion, mean imputation, or more advanced techniques like multivariate imputation, depending on the nature and amount of missing data. Given previous work, I do not expect missing data to be a significant issue; however, I expect some cleaning may be required with respect to player names across various sources.

Outliers, if present, could distort the model's predictions. Outliers will be detected using methods like the Z-score, the IQR score, or visually through box plots and treated appropriately, either by removing them or using statistical techniques to reduce their impact.

Normalisation or standardisation processes will be applied to scale numeric features, ensuring that no particular feature dominates others due to its scale. This is particularly important for algorithms that use distance-based measures or regularisation.

Feature engineering will also form a significant part of this stage. The raw data will be transformed into meaningful features that can better represent the underlying patterns in the data to the machine learning algorithms. For instance, player performance might be quantified using metrics like goals per minute, assists per game, and so on.

Categorical variables like player positions or team names will be encoded into a format that the machine learning models can understand. One-Hot Encoding or Label Encoding methods will be used depending on the variable. An example of how this works is seen below.



(Figure 4 – An example of how One Hot Encoding works (Novack, 2020))

Finally, feature selection techniques will be used to identify the most relevant features for predicting EPL Fantasy Football Points. Techniques such as Recursive Feature Elimination (RFE), Lasso Regularization, or even Tree-based methods (like Random Forest or XGBoost) might be used to rank features based on their importance. This not only helps to reduce the dimensionality of the dataset, but also aids in improving model interpretability and reducing overfitting.

The outcome of the pre-processing stage will be a clean, transformed, and reduced dataset ready to be used in the model training phase.

3.3 Training and Testing

Once the pre-processing is complete, we divide the processed dataset into two parts; training and testing sets. The training set is used to train our machine learning models, while the testing set evaluates how well they perform. Typically we allocate around 80% of the data for training and 20% for testing. However, we can adjust this split if necessary during our project.

Since our research deals with time series data (in this case, football matches occurring in a sequence), we need to take an appropriate approach to split the data. Namely, we use a special technique called "walk-forward" validation or time series split. This helps us avoid any data leakage and ensures our model can accurately predict events.

In walk-forward validation, we initially train our model on a small portion of the data. Then using that trained model, we make predictions for the period. Afterwards, we expand our window of time to include that period and retrain the model. We repeat this process until we reach the end of our dataset. This method allows us to predict points effectively while helping prevent any errors from "peeking into the future", which can occur when randomly splitting time series data. (Time Series - Walk Forward Validation,

n.d.) During the training process, we will adjust the models' hyper-parameters using techniques like grid search or random search to find the settings for the model. To prevent overfitting and ensure a reliable model, we will employ cross-validation methods that involve evaluating the model with different subsets of the training data.

In essence, our training and testing approaches are designed to produce adaptable predictive models. Our goal is to create models that not only perform well on historical data but also have the ability to make accurate predictions on future unseen data.

3.4 Modelling

We move into the modelling phase with the dataset prepared and divided into training and testing subsets. This research focuses on exploring various machine learning algorithms to identify the one that offers the most accurate predictions for EPL Fantasy Football Points.

We will consider different types of models ranging from linear models, ensemble methods, to advanced deep learning models.

- **Linear Regression**: As a baseline, we start with Linear Regression due to its simplicity and interpretability. It assumes a linear relationship between the input variables (features) and the single output variable (Fantasy Football Points).

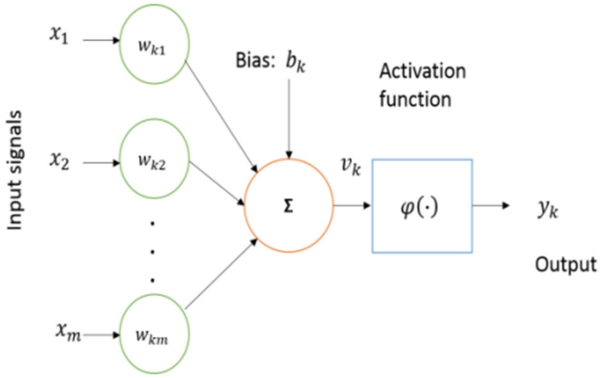
- **Decision Trees and Random Forests**: Decision Trees are a type of model that makes predictions by learning simple decision rules from the data features. They are easy to understand and interpret. Random Forests, an ensemble of Decision Trees, are used to improve the model performance and control overfitting.

- **Gradient Boosting Machines (e.g., XGBoost, LightGBM)**: These are another type of ensemble machine learning algorithms that are known for their efficiency and performance. They build an ensemble of weak prediction models, typically decision trees, in a stage-wise fashion where each subsequent model attempts to correct the errors of the previous one.

- **Deep Learning Models (e.g., LSTM, GRU)**: Given the sequential nature of the data, Recurrent Neural Networks (RNNs) like LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) might be used. These models are designed to

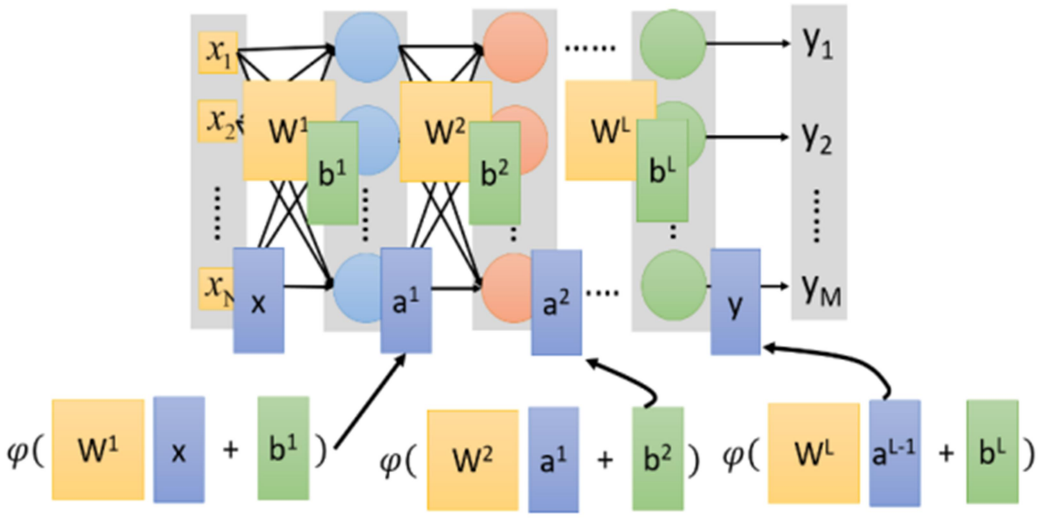
recognize patterns across time, which could potentially make them more effective for this project; particularly given that their effectiveness has also been demonstrated in some of the papers discussed in my Literature Review.

An Artificial Neural Network consists of several neurons working to give some output.



(Figure 5 – Basic model of a neuron)

The input signals are provided by the testing data we acquire from the previous stage, then some weighting is applied to these inputs and finally an activation function determines the output from the neuron. Many of these are then combined into a Neural Network.



(Figure 6 – Diagram of an Artificial Neural Network (ANN))

ANNs or general NNs consist of Multilayer Perceptron's (MLP) which contain one or more hidden layers with multiple hidden units (neurons) in them. (Zahangir Alom et al., 2018) This is shown above in Figure 6.

Each model will be trained using the training data. We will use grid search and cross-validation to tune the hyper-parameters of each model and find the optimal settings.

The choice of these models allows us to explore the prediction problem from different perspectives - linear models offer simplicity and interpretability, tree-based models provide a balance between interpretability and performance, and deep learning models might capture complex, temporal patterns in the data.

3.5 Evaluation

The final phase of the methodology involves the evaluation of the machine learning models developed in the previous stage. This process involves applying the trained models to the test set and comparing the predicted EPL Fantasy Football Points to the actual points.

The performance of the models will be evaluated using the following metrics:

- **Mean Absolute Error (MAE)**: This metric calculates the average absolute difference between the predicted and actual values. It provides a straightforward measure of prediction error rates.

- **Root Mean Squared Error (RMSE)**: RMSE measures the average squared difference between the predicted and actual values. It gives a higher weight to large errors, meaning it's more useful when large errors are particularly undesirable.

- **R-squared (Coefficient of Determination)**: This statistic indicates the proportion of the variance in the dependent variable (Fantasy Football Points) that is predictable from the independent variables (player and team features). It provides a measure of how well the model fits the data.

These metrics collectively give a comprehensive view of the model's performance, highlighting both the average error rate and how well the model's predictions fit the actual values.

Furthermore, a model's performance can be visualised using plots such as residual plots and actual vs. predicted plots. This helps in understanding the model's performance in a more intuitive manner.

We will also perform a comparative analysis of the models based on their performance metrics. The model with the best overall performance, considering the balance between prediction accuracy and model complexity, will be selected as the final model. Additionally, Model performance will be evaluated for different types of players (e.g., forwards vs. defenders), different teams, and different time periods to understand the model's strengths and weaknesses, and identify areas of potential improvement.

By following this robust evaluation process, we aim to ensure the selection of a high-performing, reliable model for predicting EPL Fantasy Football Points.

4 Implementation

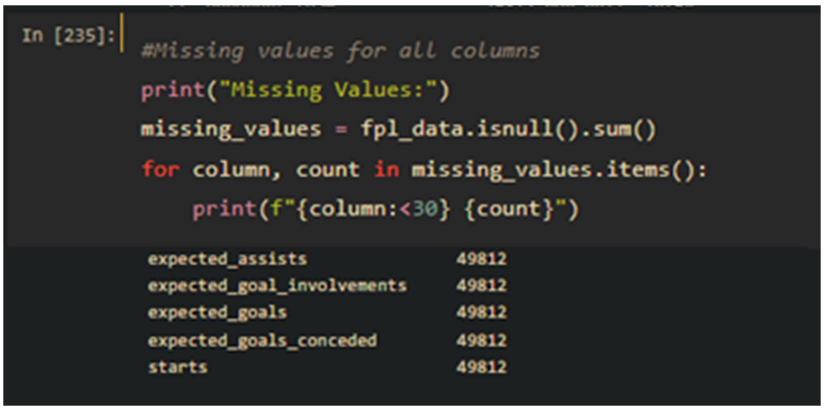
In this section, results obtained from deploying the methodology are described in detail. A series of models were produced, their efficacy measured and the results are provided. Comparison of the models to each other will be conducted in my Conclusions.

A full account of the code can be found in Appendix A at the end of this report.

4.1 Data Collection and Preparation

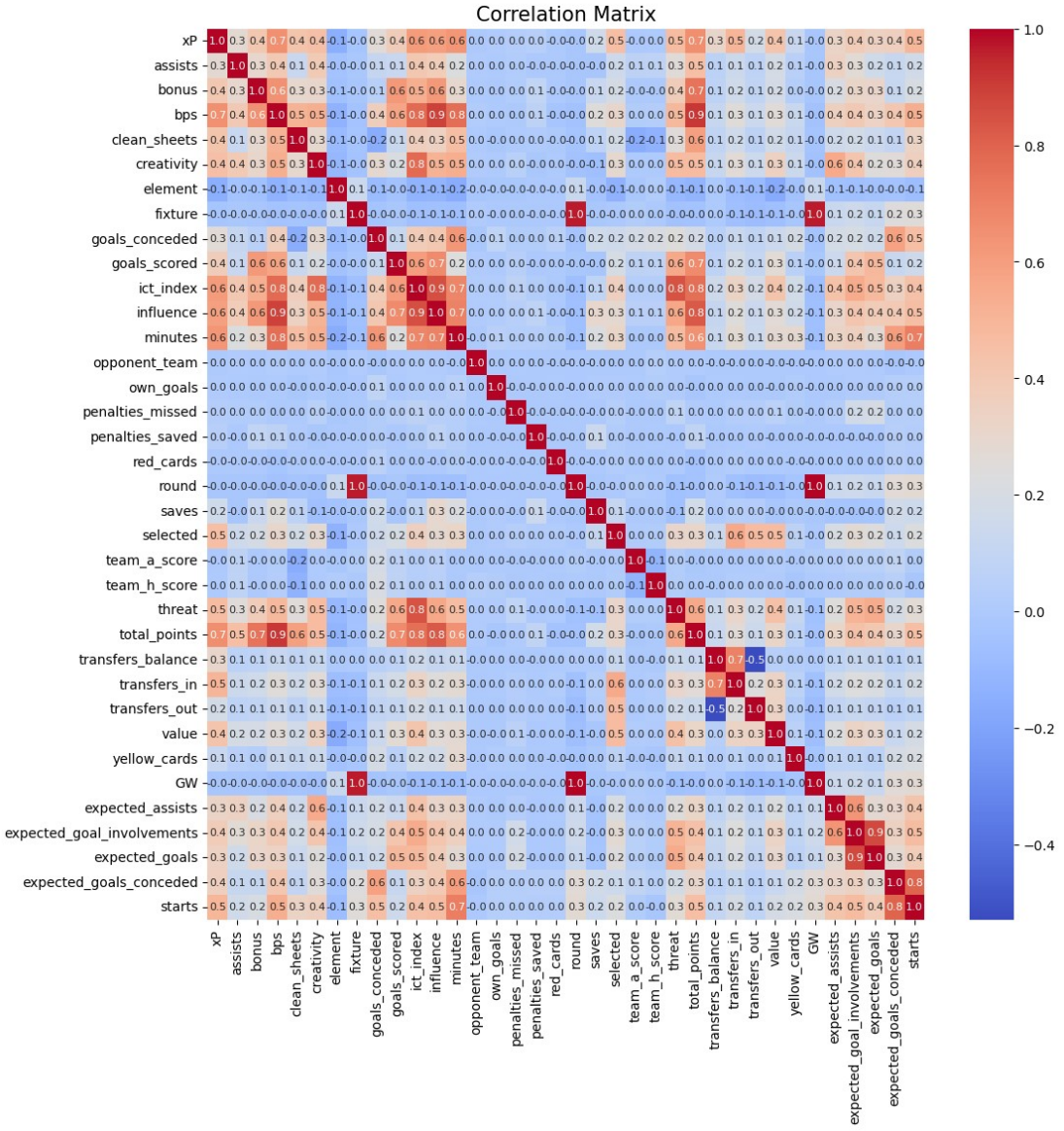
The first stage in our process was collecting the data and preparing it for processing. Data was downloaded from Vaastav's Fantasy Premier League Github Repository and from Kaggle for the seasons 2020/21, 2021/22 and 2022/23 (representing the last three seasons of the EPL).

We started with the FPL dataset, and upon visual inspection of the data, it was quick to note that the "Expected" values were not present for all seasons; in fact, only the most recent season captured this data fully (see Figure 7 below). These expected variables would be instrumental in prediction given that they are somewhat predictive in themselves; however, a single season of data is not a lot of data points to work with. There are 38 Game Weeks in a single season; as such, we would be training models on only 38 points per player; this needs to be bigger. Therefore the decision was made to exclude these columns from our analysis.



(Figure 7 – Snippet from Python showing missing values for FPL data)

Given the sheer number of variables, a correlation matrix is somewhat challenging to read; however, not entirely without merit. We can see some clear correlations we should beware of. Some variables tell us the same information as others. For example, 'Round' and 'GW' are essentially the same metric.



(Figure 8 – Correlation Matrix of FPL dataset)

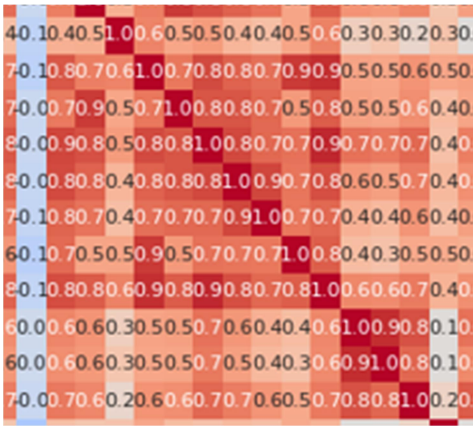
Next, we ingest the FIFA Data. We are less concerned with variables in this data set, and generally, the fields we expect to be useful do not contain NULL values. However, it is worth noting that "Club" contained several blanks, as this proved problematic in later stages (See Figure 9).

```
In [241]: #Missing values for all columns
          print("Missing Values:")
          missing_values = fifa_data.isnull().sum()
          for column, count in missing_values.items():
              print(f"{column:<30} {count}")

          Missing Values:
          ID                             0
          Name                           0
          Age                            0
          Photo                          0
          Nationality                    0
          Flag                           0
          Overall                        0
          Potential                      0
          Club                           800
```

(Figure 9 – Snippet from Python showing missing values for FIFA data)

Given the sheer number of variables, a correlation matrix was again somewhat difficult to read; however not entirely without merit. We can see there is a lot of conflation of variables; this is to be expected as the "Overall" statistics are clearly some combination of the more intricate ones. Statistics like "Curve" obviously play some part in things like "Free Kick Ability". A small portion of this matrix is provided below in Figure 10 to demonstrate how strongly related some variables were.



(Figure 10 – Crop of a section from Correlation Matrix for FIFA dataset)

Merging the Datasets

Now that the data sets are ingested, we need to combine them. We found From the Literature review that Kirstinsson encountered several data cleansing issues while trying to match player names between the FIFA and FPL Datasets. As such, we need to inspect the various names in each set and see what string manipulations we can employ such that they will match and allow us to join the data successfully.

As part of this step, it became apparent that to join the data effectively, we needed to match the FIFA data for the corresponding year and not just use the information from the latest game on player statistics.

The first problem we encountered was differences in Club names across the two data; we used a simple dictionary mapping to alter these in order to facilitate the joins properly. The dictionary was manually created since there are only 20 teams in the EPL in any one season and only 25 different teams across the three seasons we've gathered. A snippet of the code used to do this is shown below in Figure 11.

```
#Create the mapping to replace the club names
name_mapping = {
    'Brighton & Hove Albion': 'Brighton',
    'West Ham United': 'West Ham',
    'Sheffield United': 'Sheffield Utd',
    'Everton': 'Everton',
    'Fulham': 'Fulham',
```

(Figure 11 – Snippet of code for mapping team names)

Now we move onto the most difficult part of the joins; the formatting of the names is very different across both datasets. The FPL data is primarily a player's full legal name. Whilst the FIFA data is occasionally the nickname a player goes by. For example, "José Ignacio Peleteiro Romallo" is the full legal name of now-retired Spanish footballer "Jota" who played for Aston Villa in 2020/21, and these are the names used in the two sets respectively.

As such, we need to perform our joins on several fields, Name, Team and Year – this should ensure that we have the correct player's statistics. However, this means we need some method for joining seemingly incompatible names together. To do this, we

took a three-tiered approach using "fuzzy" logic, followed by a final manual process for any discrepancies observed.

Fuzzy Join One

Using the fuzzywuzzy package in Python, we create a function which can iterate over the rows in our dataset. For this first stage, it will only match where the team and year are identical and where the similarity score is over 80; this should ensure that we do not over-fit the data with bad matches.

Every match is then stored in a dictionary that we will use for all stages, "name_matches", and we will use this same dictionary to test whether we have already matched a name, allowing us to remove some of the computational strain.

Fuzzy Join Two

For the next stage, we ignore the year and any names we have previously matched. This means that providing the teams match, it will look for a player name that is as close to a match as possible (again using a similarity score of over 80 to avoid over-fitting).

During this stage, I noted from visual inspection that some players were missing because their team name was NULL. As such, another sub-stage was added to this. This extra step allows the function to fit both instances where the name matches or where it is NULL in the FIFA Data.

These matches are then also added to the dictionary for later use.

Fuzzy Join Three

Finally, in our fuzzy logic steps, we get less robust. We simply take any names which have not yet been matched in and let fuzzywuzzy match any entry across the entire FIFA dataset where the similarity score is over 45.

This step led to many bad matches, but also to some that would have been missed without it, but all names were matched in some fashion (see Figure 12)

```
#Check how many names are still missing
print('Current Missing Values: ',len(fpl_data[fpl_data['processed_name'].isna()]))
```

```
Current Missing Values:  0
```

(Figure 12 – No missing values in our processed_name column)

It was apparent at this point that a final manual stage would be needed to amend the incorrect matches.

Manual Step

The name_matches dictionary was exported to CSV and a manual inspection of the output was undertaken. Some 115 names were identified as possibly wrong or needing correction. A small sample of this is provided in Figure 13 below.

FPL Name	FIFA Name
Bamidele Alli	D. Alli
Bernardo Fernandes da Silva Jur	Bernardo
Bernard AnÃ-cio Caldeira Duarte	Bernard
Gabriel MagalhÃ£es	Gabriel
Matthew Longstaff	S. Longstaff
Oluwasemilogo Adesewo Ibidaı	S. Ajayi
Ã‡aglar SÃ¶yÃ¼ncÃ¼	C. SÃ¶yÃ¼ncÃ¼
Bernardo Mota Veiga de Carvalh	Bernardo Silva
James Garner	D. James
Thakgalo Leshabela	20Â K. Leshabela
Pelenda Joshua Dasilva	J. Dasilva
Aaron Ramsey	J. Ramsey
Bernardo Veiga de Carvalho e Si	Bernardo Silva
Stefan Ortega Moreno	M. Moreno
Kadan Young	A. Young
Lewis Miley	J. Lewis
Charlie Robinson	A. Robinson
Ayotomiwa Dele-Bashiru	T. Dele-Bashiru
Bernardo Fernandes Da Silva Jur	Bernardo
Luke Harris	J. Harris
Sebastian Revan	D. Revan
Ahmed El-Sayed Hegazy	I. Ahmed
Jonathan Castro Otto	V. Castro
Johann Berg Gudmundsson	M. Berg

(Figure 13 – Manual dictionary inspection)

Once these names were checked and amended appropriately, a final step of amending the name_matches dictionary was implemented within the Python code. In some cases,

names were removed entirely because no appropriately accurate match could be determined.

Finalisation

Once the name mapping was applied to the dataset, we were ready to join our sets together. Again, this was conducted in two stages. Firstly all rows where processed_name, team and year matched were joined together on these columns. Secondly, if a join was not found, the remaining data was joined on processed_name and year only. This allowed us to properly account for cases where players had the same processed_name.

Following the joins, there was some very minor duplication, specifically concerning a single player who appeared to be assigned multiple "position" values. A simple approach was taken to deal with this; keeping the first entry for each duplication.

The data was then exported to CSV for later use in our Data Processing stage.

4.2 Data Processing

The next stage in our project was data processing. We are trying to get the data into a format suitable for the machine learning methods we wish to compare and ultimately use this transformed dataset for training and testing these models.

Initially, we inspect the data and notice several columns which can be removed from the outset: processed_name can be dropped as it was only required to perform our joins; team_y, team_x, and marking can all be removed as they only contain NULL values; name_y can also be removed as it was created as part of the joining process.

Additionally, as previously stated, "expected" columns are not populated for all seasons. As such, we shall remove all of these columns. Likewise, "starts" is not populated for all seasons and can also be removed.

There are several columns which offer little or no predictive value simply from understanding the information they provide. These have also been removed. The Python code to remove some of these columns is shown below in Figure 14.

```
#There are clearly several extra columns we can remove, name
# ID, Joined, Club Logo, Flag, Photo, Release Clause, Contr
#These all should have no value in future points prediction
df.drop(columns=['Loaned From',
                 'Kit Number',
                 'Unnamed: 0',
                 'ID',
                 'Joined',
                 'Club Logo',
                 'Flag',
                 'Photo',
                 'Release Clause',
                 'Contract Valid Until'], inplace=True)
```

```
#From inspection we can see several columns that c
# or at the very least will be explained by other
# and Best Position can also all be removed
df.drop(columns=['Nationality',
                 'Body Type',
                 'Real Face',
                 'Position',
                 'Best Position'], inplace=True)

#Additionally kickoff_time should be a datetime el
# for simplicity
df.drop(columns=['kickoff_time'], inplace=True)
```

(Figure 14 – Code snippets for the removal of unused columns)

Data Type Conversions

During processing, we noticed several columns in a format unsuitable for their meaning, i.e. numerical values stored as objects. This was due to some mixed type values in their contents.

The offending columns were Height, Weight, Value and Wage. The Value and Wage columns contained entries formatted as €3m and €600k; the Height column included entries formatted as 5'9 and 180cm; the Weight columns contained entries formatted as 170lbs and 80kg. Several functions were created to solve these issues and convert the data into the correct numerical types. An example can be seen in Figure 15.

```
#Let's start with Height
#Function to convert height in feet and inches to centimeters
def convert_to_cm(height):
    if pd.notna(height) and "'" in height:  # If height is not NULL and in feet and inches format
        feet, inches = map(int, height.split("'"))
        total_inches = (feet * 12) + inches
        return total_inches * 2.54
    elif pd.notna(height) and "cm" in height:  # If height is not NULL and in centimeters format
        return int(height.replace("cm", ""))
    else:
        return None
```

(Figure 15 – Conversion function for Height column)

Handling Missing Values

Missing values can be tough to handle; several methods were tried and tested, covering deletion, mean imputation, multivariate imputation and finally, KNN (k-Nearest Neighbours) imputation. Upon inspection, a substantial portion of the data had at least one missing value in some column of the data. This meant we immediately ruled out deletion as a method as it would reduce our dataset too significantly. Mean imputation was likely to cause issues as many of the data items had considerable spread in their values, so a simple average may not be sufficient. Multivariate imputation also appeared, prima facie, to not provide reasonable estimates as the variables are perhaps not closely related enough.

Ultimately this led us to use KNN imputation (see Figure 16 below) as it provided a more sophisticated method than just using something broad like the mean, while still seeming to give reasonable values. Unfortunately, it isn't perfect as we have some strong correlations between some variables, as we saw previously.

```
#Method 4: KNN Imputation - replacing missing values using the k-Nearest Neighbours approach
knn_imputer = KNNImputer(n_neighbors=5)  #Use 5 nearest neighbours
df_knn_imputed = df_numeric.copy()
df_knn_imputed[df_numeric.columns] = knn_imputer.fit_transform(df_numeric)
```

(Figure 16 – KNN imputation)

Normalisation

This process was simple. Given the spread of the data, we normalised the data using a standard scaler to ensure that particularly large variables do not carry any excess weight. Figure 17 shows the code for this.

```
#Normalise data using standard scaler
scaler = StandardScaler()
df_normalized = pd.DataFrame(scaler.fit_transform(df.select_dtypes(include=[np.number])),
                             columns=df.select_dtypes(include=[np.number]).columns)
```

(Figure 17 – Normalisation using standard scaler)

Encoding Categorical Variables

Following normalisation, we undertake encoding of our categorical variables. There are two preferred methods, One-Hot Encoding should be used for nominal data (i.e. that which has no order) and Label Encoding should be used for those which have some order.

Every categorical variable in our dataset is nominal; as such, we can use One-Hot Encoding across all of them. Additionally, now is an excellent time to convert our name_x column into some numerical ID. We will store these conversions in some dictionary for later use. See Figure 18 for how this was conducted.

```
#Now is a good time to convert name_x into some ID (since it avoids our normalisation)
#We will do this using factorize()
df_categorical['ID'] = pd.factorize(df['name_x'])[0]

#In order to let us convert these back from our final models, we will create a dictionary also
id_name_dict = dict(zip(df_categorical['ID'], df_categorical['name_x']))
#We can now also drop the name_x column
df_categorical.drop(columns=['name_x'], inplace=True)

#Perform one-hot encoding on the categorical columns
one_hot_encoded = pd.get_dummies(df_categorical)

#Combine the one-hot encoded and normalised dataframes
df_processed = pd.concat([df_normalized, one_hot_encoded], axis=1)
```

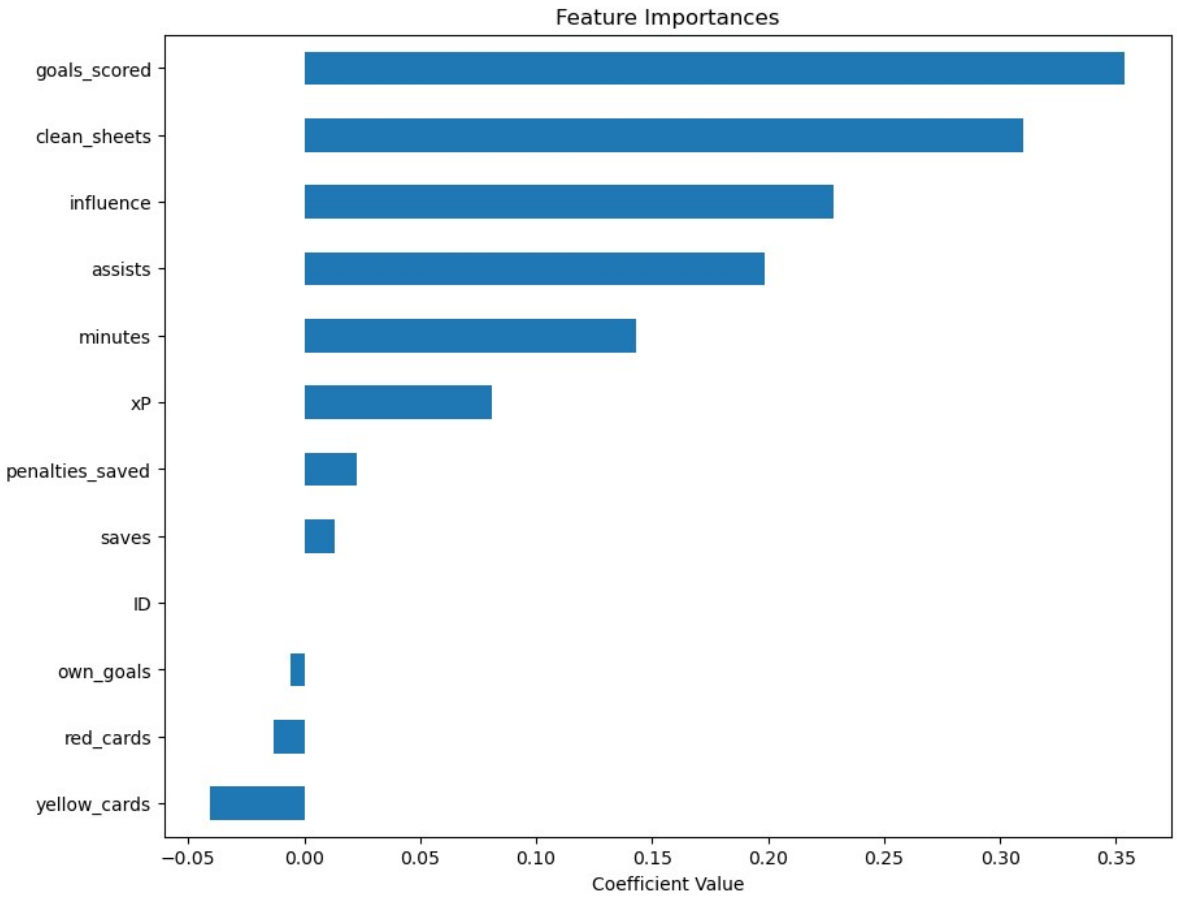
(Figure 18 – One-Hot Encoding & ID creation)

Dimensionality Reduction

We are now ready to perform our dimensionality reduction. Random Forest Regressors can handle numerical and categorical data and do not require data to be scaled. They also provide a very easy-to-interpret measure of feature importance. As such, we used this as a first attempt to give some base to compare.

However, since all variables have been scaled and One-Hot Encoding has been used on categorical variables, we should consider using Lasso regularisation for feature selection. Lasso (Least Absolute Shrinkage and Selection Operator) has the ability to shrink the coefficients of less important features to precisely 0, effectively performing feature selection.

Using Lasso proved very effective, leaving us with 12 significant features for use in our final models; Figure 19 provides a chart of these features and their relative importance. This reduced data set was then exported along with our target variable "total_points", ready for our machine learning models to be applied.

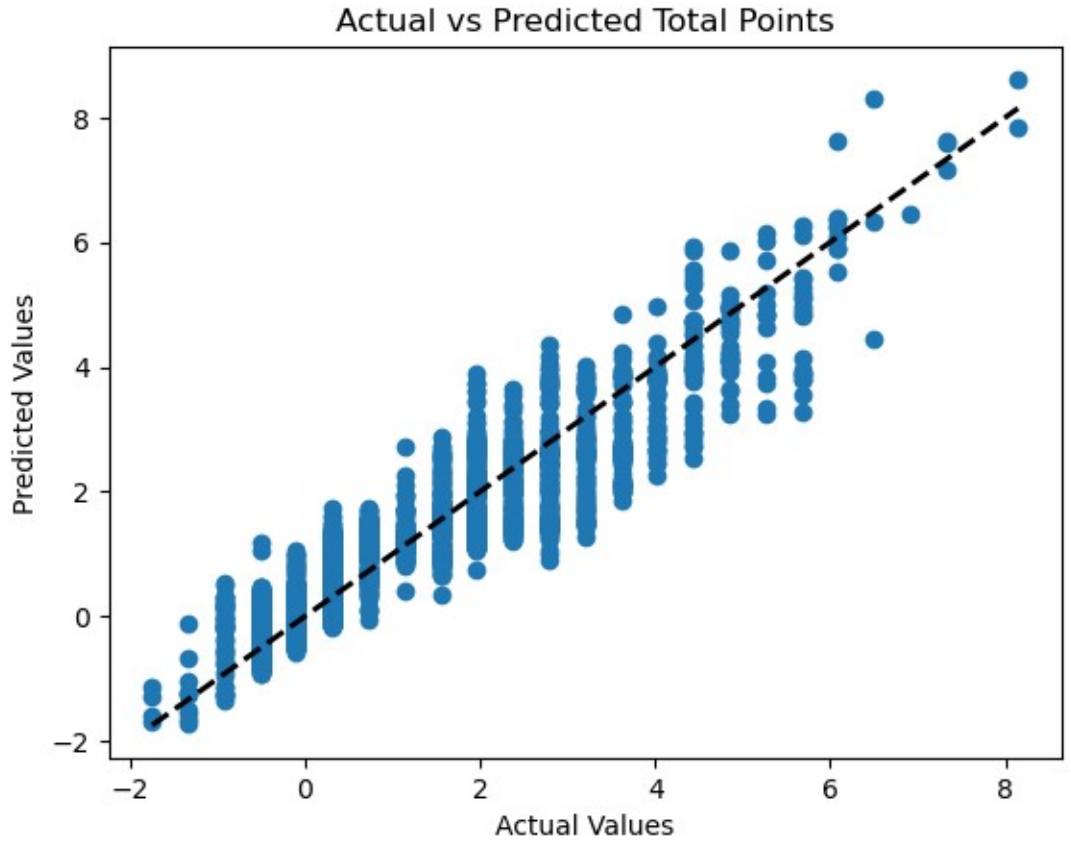


(Figure 19 – Lasso Feature Importances)

4.3 Test and Train

Linear Regression

Linear Regression is our most basic model and will be a benchmark for the other models we produce. To keep this model as simple as possible, we will not use walk-forward validation either. This should ensure it is a good base level for later comparison.



(Figure 20 – Linear Regression model Predicted vs Actual)

In Figure 20, we can see some heteroscedasticity, which means our model's performance is varying across different ranges. Perhaps we are not capturing some non-linear relationship. However, when we test the total points for a strong player like Kevin De Bruyne, it is only 11 points off the actual for that season (183), so anecdotally; we have a reasonable model even at this early stage.

Mean Absolute Error:	0.15293146117847595
Root Mean Squared Error:	0.2933898727682214
R^2:	0.9024352940157163

The linear regression model seems to perform quite well, with a high R-squared value of 90.24%, indicating a good fit to the data. The MAE (0.15) and RMSE (0.29) values

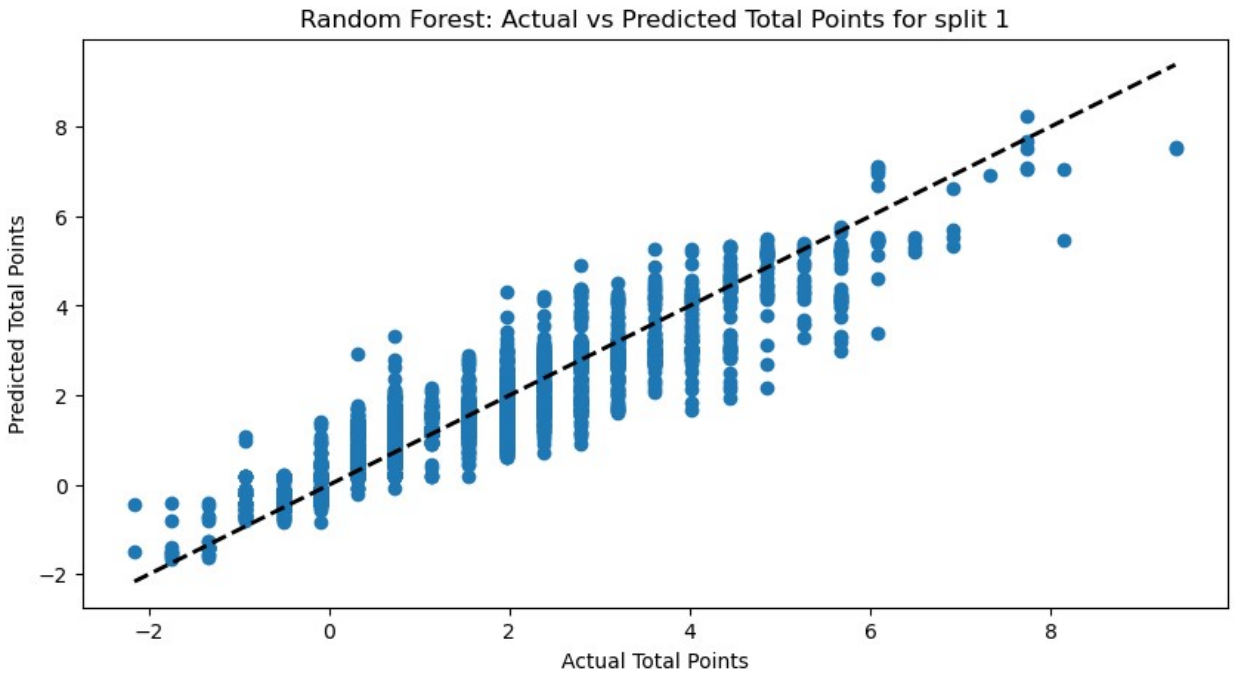
provide a sense of the average error magnitude, with the RMSE giving more weight to larger errors. The average error is quite reasonable given that the spread of points across the entire dataset for a given week is between -4 and 23 points.

Random Forest

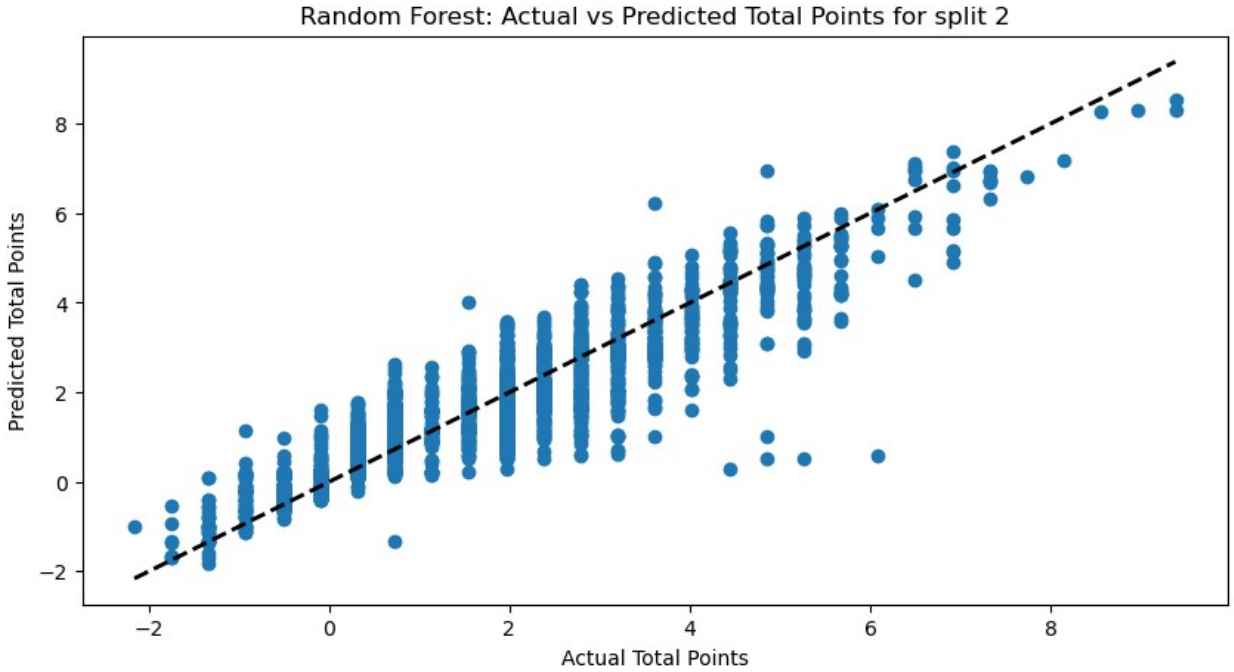
With Random Forest, we implement our first instance of walk-forward validation. Given that the data is across three seasons, it seems logical that there could be different optimal models for each season. As such, we have used TimeSeriesSplit from sklearn to perform these splits and use Grid Search to find the best parameters for our model.

Once grid search finds the best model for a given split, it stores this model for later use.

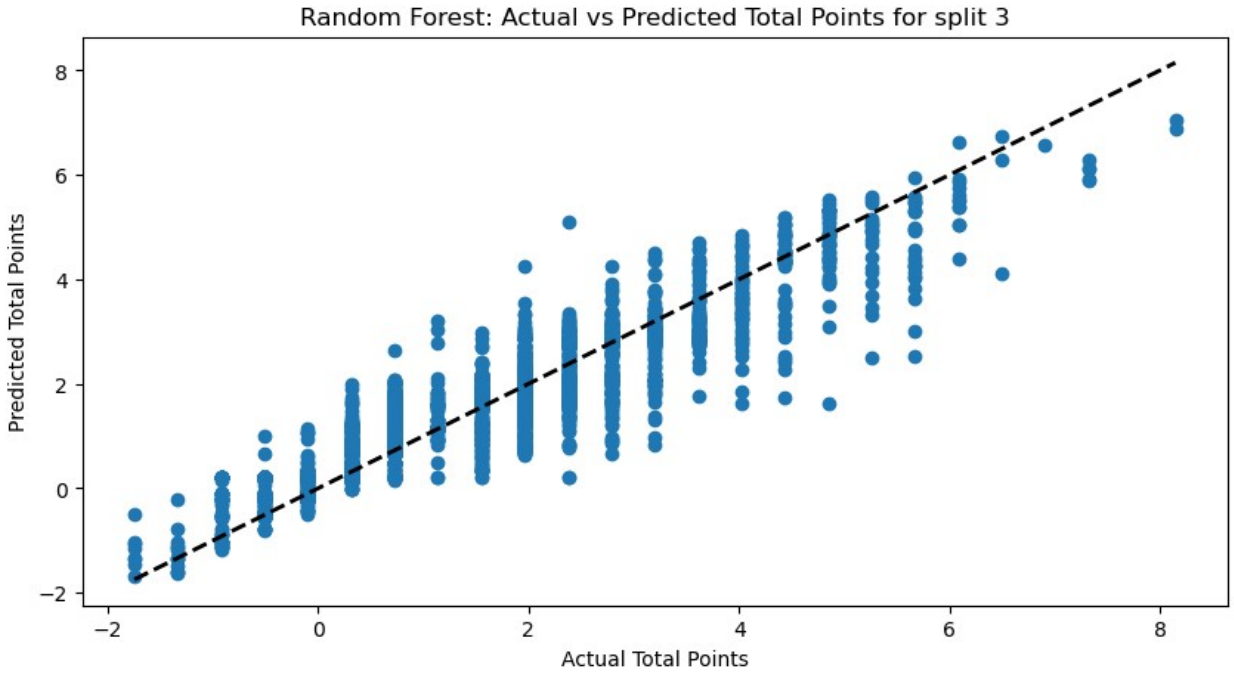
The following plots relate to each split (Figures 21, 22 and 23)



(Figure 21 – Random Forest Actual vs Predicted for split 1)



(Figure 22 – Random Forest Actual vs Predicted for split 2)



(Figure 23 – Random Forest Actual vs Predicted for split 3)

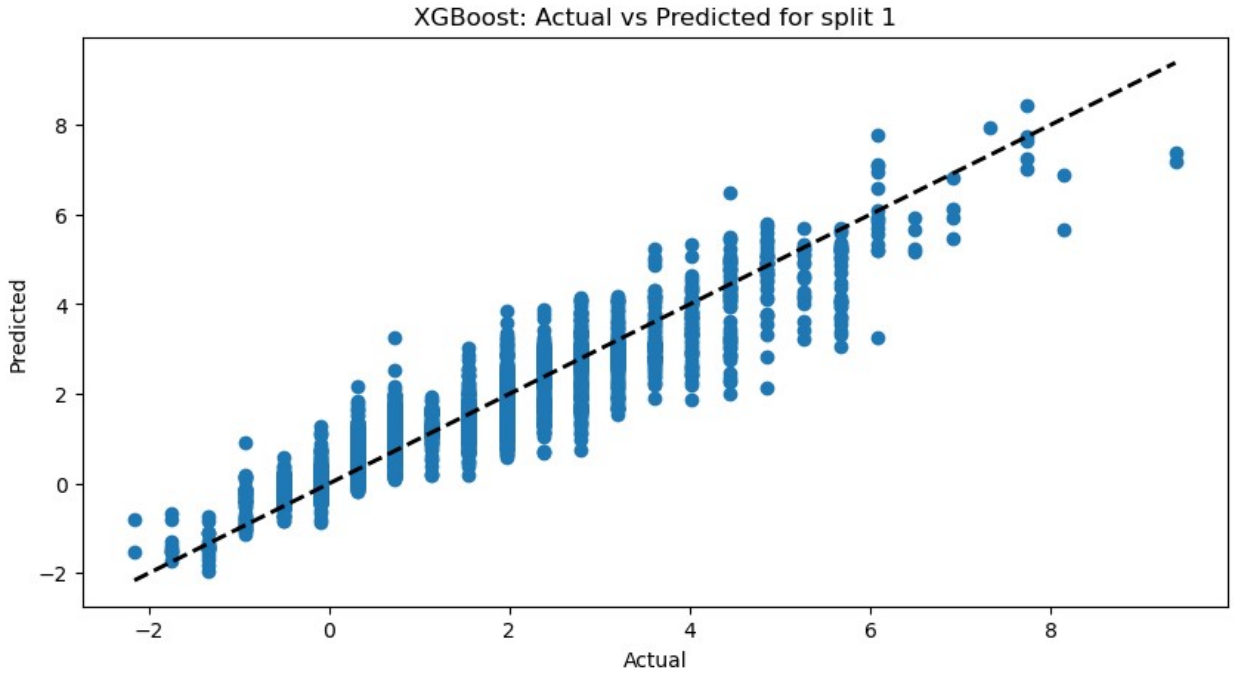
We still see some heteroscedasticity in all three of our models even when splitting across time sections. This means our models' performances are varying across different ranges. There may be some other subgroups which should be captured in future modelling to provide better results. When we test the total points for Kevin De Bruyne, we can see it is further out, now almost 21 points off the actual for that season (183).

	Split 1	Split 2	Split 3
MAE:	0.10791334494547196	0.10871365940030689	0.0943156942223728
RMSE:	0.26898244497582385	0.28531231847999033	0.24973234703259084
R^2 :	0.9293637599319641	0.91917786434858	0.9308520053699395

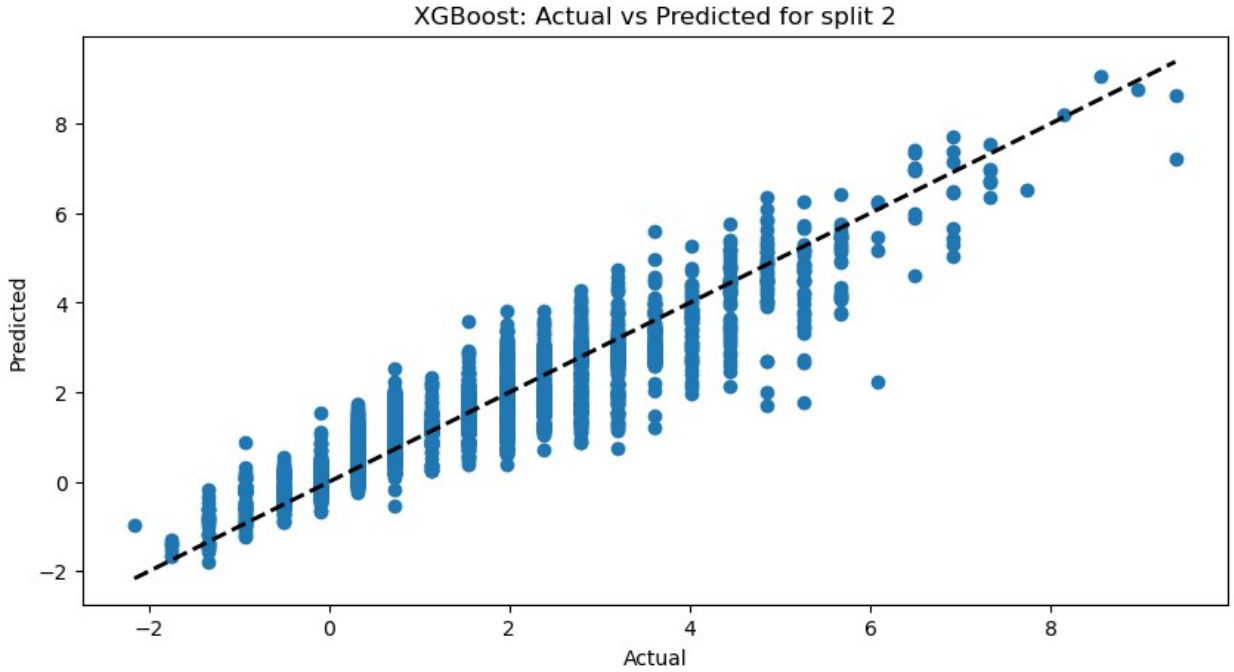
The Random Forest models generally appear to perform very well, with our final split (the latest season) capturing over 93% of the variability in the target variable. The error metrics, both MAE (0.094) and RMSE (0.249) are quite low, suggesting that, on average, the model's predictions are close to the actual observations.

XGBoost

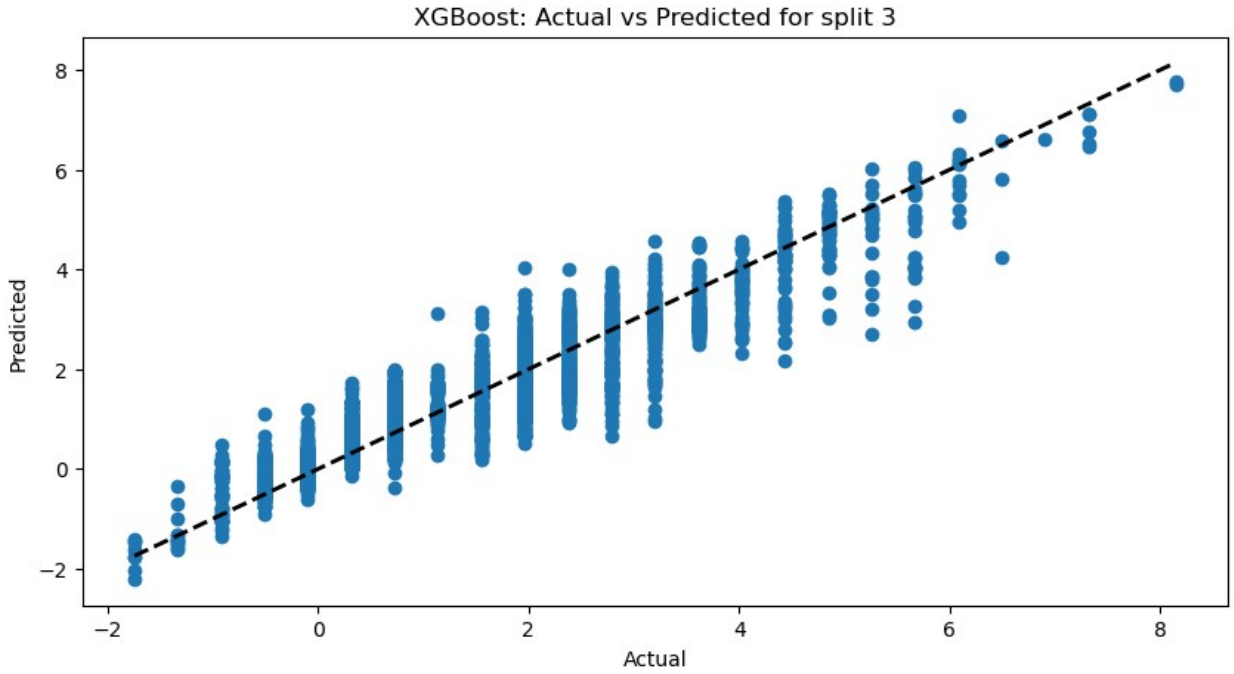
As with Random Forest, we use TimeSeriesSplit and Grid Search to determine the best models. The following plots relate to each split (Figures 24, 25 and 26)



(Figure 24 – XGBoost Actual vs Predicted for split 1)



(Figure 25 – XGBoost Actual vs Predicted for split 2)



(Figure 26 – XGBoost Actual vs Predicted for split 3)

Again, we see some heteroscedasticity in our models even when splitting across time sections. This means our models' performances are varying across different ranges. As before, it seems likely there are some other subgroups which should be captured in future modelling to provide better results. Given XGBoost is not strict with its assumptions like Linear Regression, we expect the problem is not caused by any non-linear relationship not being captured.

When we test the total points for Kevin De Bruyne, we can see it has slightly improved, now 19 points off the actual for that season (183).

	Split 1	**Split 2**	**Split 3**
MAE:	0.10859083451831386	0.11007702588510027	0.09210786188067581
RMSE:	0.2640091969739644	0.27514456944621823	0.24067897747893754
R^2 :	0.931951616582233	0.9248357778727221	0.935774675457326

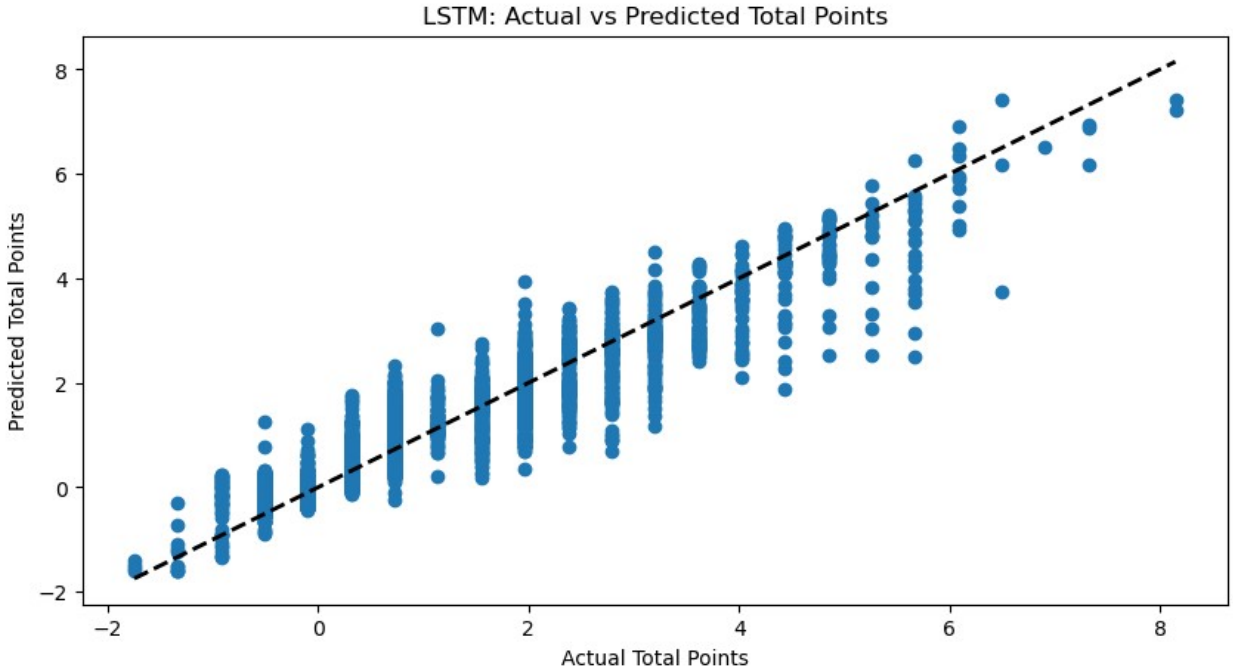
The XGBoost models seem to perform very well, in our final model for split 3; it captures nearly 93.58% of the variability in the target variable, and both error metrics, MAE (0.092) and RMSE (0.240), are relatively low. This suggests that, on average, the model's predictions are close to the actual values.

LSTM Deep Learning

Finally, we delve into some deep learning. Due to computational constraints, we were unable to perform this method across our time series split as with the two previous models. This is something I would like to revisit in my Further Work.

Again due to the heavy computational load, we rely entirely on the model hyper-parameters encountered in our Literature Review. Namely, those used by Kristinsson in his work related explicitly to Deep Learning models. These parameters are below:

Batch:	33
Optimizer:	Adam
Learning rate:	0.0004
Dropout rate:	0.5
Epochs:	250
Units:	256



(Figure 27 – LSTM Actual vs Predicted)

For our final model, we see still some heteroscedasticity, although perhaps slightly reduced. It is now almost certain there are some other sub groups or clusters which should be captured in future modelling to provide better results.

When we test the total points for Kevin De Bruyne, we can see it has slightly improved again, now close to 16 points off the actual for that season (183).

MAE:	0.10900829042540931
RMSE:	0.2478288953297771
R^2 :	0.9303844297979336

The LSTM deep learning model appears to perform strongly based on our metrics. It accounts for about 93.04% of the variability in the target variable, and the error metrics, MAE (0.109) and RMSE (0.248), suggest relatively close predictions to the actual values, on average.

However, LSTMs and deep learning models, in general, have high capacity and can easily overfit to training data. As such, we should take these results with caution.

5 Conclusions

We have attempted to tap into the power of machine learning to try and provide some actionable insights in the realm of Fantasy Football. This wasn't just a data-driven task; it was about blending passion for the sport with analytical rigor. With an almost overwhelming wealth of data at our disposal and modern techniques in our toolkit, we attempted to valid and maybe build on what others have done. As we conclude this project, let's revisit our models, discuss their efficacy, and contemplate the next steps.

5.1 Model Comparison

Since the most recent season is potentially the most relevant, let us use this as the basis of our comparison with regard to the time-split models. Below we see a comparison of the metrics we gathered.

	Regression	**Random Forest**	**XGBoost**	**LSTM**
MAE:	0.15293	0.09432	0.09211	0.10901
RMSE:	0.29339	0.24973	0.24068	0.24783
R^2 :	0.90244	0.93085	0.93577	0.93038

Based on our metrics, it would appear that XGBoost is the best-performing model, with the best values across all categories. Our next best-performing model is close between LSTM and Random Forest, the primary difference being that LSTM is performing better on RMSE. This suggests the LSTM model is slightly better as there are likely fewer large errors in the predictions. Finally, our most basic model, Linear Regression, performs the most poorly, as we would expect. However, simplicity is not always bad. It is still performing reasonably considering it is such a simple model and is very quick to deploy. There may be some cases where a more rapidly deployable model is preferable, and this would certainly beat out the labour-intensive LSTM model were it necessary.

5.2 Future Work

Our study has just scratched the surface, and there remains a plethora of avenues to explore further. With the rapidly evolving nature of data science, there's always room for refinement, expansion, and new approaches. I list below some potential avenues for further investigation.

5.2.1 Incorporation of “Expected” Data from Additional Seasons

One of the potential enhancements to the dataset would be to include "Expected" data from more football seasons. This data type, which gives a probabilistic forecast of a player's performance based on various match events, can provide invaluable insights that go beyond traditional statistics. By integrating data from multiple seasons, we can capture long-term patterns, identify consistent performers, and enhance the robustness of our model, leading to more accurate and reliable predictions.

5.2.2 Comprehensive Examination of Missing Value Imputation

Handling missing values is a fundamental step in data pre-processing. While we employed certain imputation techniques in our study, a comprehensive analysis of different imputation methods could lead to significant improvements in model performance. Exploring other advanced methods like iterative imputer or using deep learning techniques might prove beneficial. It is also possible a simpler technique might yield better results. Understanding the nuances of how each method impacts the final model will offer deeper insights into the reliability of our analytics.

5.2.3 Hyper-parameter Optimisation

Although we did some preliminary hyper-parameter tuning, spending more time on this front has the potential to produce better results. Leveraging methods like Bayesian optimisation, genetic algorithms, or more extensive grid searches can help to pinpoint the optimal configuration of parameters for our models, ensuring maximum performance. Such meticulous tuning often uncovers subtle nuances in models that can lead to significant improvements in predictions.

5.2.4 Dive Deeper into Recurrent Neural Networks (RNNs)

Given the sequential nature of football data, with matches unfolding over time and players' performances influenced by past events, RNNs, especially their advanced counterparts like LSTMs and GRUs, hold significant promise. A focused study on these neural architectures, tailor-made for time series data, might unravel patterns that traditional machine learning algorithms might miss. By dedicating resources specifically to the exploration and optimisation of RNNs, there's potential for a breakthrough in the prediction accuracy and insights derived from the data.

6 Critical Evaluation

6.1 Evaluation of Project Conduct and Appropriateness of Methods

Our journey in predicting Fantasy Football Points using machine learning was marked by methodical planning and strategic decisions. Regular meetings with my project supervisor proved useful and careful time management was key to remaining on track with such a short timescale. However, upon reflection, certain elements deserve a closer look.

The chosen algorithms and methods, while state-of-the-art, may not always capture the intricate, unpredictable dynamics of football matches. We employed a mix of classical and modern machine learning techniques, but the inherent unpredictability of sports outcomes means there's always room for improvement. The key takeaway? The best statistical methods can't always guarantee success in the real world. Given more time, integrating more sophisticated models or blending different predictive algorithms might lead to improved results.

6.2 Consideration of Risks

The research, albeit interesting, was not without risks. The reliance on datasets from third-party platforms such as GitHub and Kaggle meant that there was an inherent risk of incomplete or biased data. This could potentially skew results or lead to overfitting in our models. Furthermore, the dynamic nature of football—with players transferring, getting injured, or hitting unexpected forms—introduces uncertainties that can greatly influence point predictions.

6.3 Legal, Social, and Ethical Issues

As Data scientists, we must recognise the importance of ethical considerations when dealing with data. The datasets employed were sourced from public platforms, ensuring no breach of proprietary data. Yet, a lingering question remains: do players and clubs have reservations about such extensive data on their performances being freely available?

From a social perspective, while the project can enhance the gaming experience for Fantasy Football enthusiasts, there's the risk of reducing the sport to mere numbers, possibly taking away its intrinsic unpredictability and charm.

Legally, data rights in the age of digital transformation can be a maze. Ensuring that data usage did not infringe on any copyrights or proprietary rights was paramount. Even as the datasets in use are public, the evolving nature of data laws means continuous monitoring is essential to stay compliant.

In conclusion, while the project brought forth intriguing insights and showcased the prowess of machine learning in sports analytics, it also underscored the importance of a holistic, mindful approach. Every data-driven prediction in sports, or any other field, is a blend of numbers, real-world dynamics, and a fair share of unpredictability. This project was not just an academic exercise but a reminder of the vast possibilities and responsibilities that come with the convergence of data science and real-world applications.

Bibliography and References

[1] Fan Favorite: The Global Popularity of Football is Rising. (2018, June).

a. *Nielsen.* *https://www.nielsen.com/insights/2018/fan-favorite-the-global-popularity-of-football-is-rising/*

[2] Future, M. R. (2023, May 30).

a. *Fantasy Sports Market Size to Reach USD 84.9 Billion, With a CAGR of 14.50% by 2032 - Report by Market Research Future (MRFR). GlobeNewswire News Room.* https://www.globenewswire.com/en/news-release/2023/05/30/2678373/0/en/Fantasy-Sports-Market-Size-to-Reach-USD-84-9-Billion-With-a-CAGR-of-14-50-by-2032-Report-by-Market-Research-Future-MRFR.html

[3] Fantasy Premier League, Official Fantasy Football Game of the Premier League. (2019). *Premierleague.com.* https://fantasy.premierleague.com/help/rules

[4] Lewis, M. (2003). *Moneyball : The Art of Winning an Unfair Game*. W.W. Norton.

[5] Ambikesh Jayal, Allistair Mcrobert, Giles Oatley, & O'donoghue, P. (2018). *Sports analytics : analysis, visualisation and decision making in sports performance*. Routledge. Copyright.

[6] Link, D. and Springerlink (Online Service (2018). *Data Analytics in Professional Soccer: Performance Analysis Based on Spatiotemporal Tracking Data*. Wiesbaden: Springer Fachmedien Wiesbaden.

[7] Yang, T.-J., & Swartz, T. B. (2021). *A Two-Stage Bayesian Model for Predicting Winners in Major League Baseball*. *2*(1), 61–73. https://doi.org/10.6339/jds.2004.02(1).142

[8] Sipko, M., & Knottenbelt, W. (2015). *MEng Computing -Final year project Machine Learning for the Prediction of Professional Tennis Matches*. https://www.doc.ic.ac.uk/teaching/distinguished-projects/2015/m.sipko.pdf

[9] Groll, A., Ley, C., Schauberger, G., Van Eetvelde, H., & Zeileis, A. (2019). *Hybrid Machine Learning Forecasts for the FIFA Women's World Cup 2019*. https://arxiv.org/pdf/1906.01131.pdf

[10] Osken, C. and Onay, C. (2022). Predicting the winning team in basketball: A novel approach. *Heliyon*, 8(12), p.e12189. doi: https://doi.org/10.1016/j.heliyon.2022.e12189

[11] Reade, J. J., Singleton, C., & Brown, A. (2020). Evaluating strange forecasts: The curious case of football match scorelines. *Scottish Journal of Political Economy*. https://doi.org/10.1111/sjpe.12264

[12] Az. (2021, June 10). *The FFS Pro Pundits*. Best FPL Tips, Advice, Team News, Picks, and Statistics from Fantasy Football Scout. https://www.fantasyfootballscout.co.uk/the-ffs-pro-pundits/

[13] Fujisaki, I., Yang, K., & Ueda, K. (2023). *On an effective and efficient method for exploiting the wisdom of the inner crowd*. *13*(1). https://doi.org/10.1038/s41598-023-30599-8

[14] Balawejder, M. (2022, February 18). *Premier League Predictions Using Artificial Intelligence*. Nerd for Tech. https://medium.com/nerd-for-tech/premier-league-predictions-using-artificial-intelligence-7421dddc8778

[15] Kristinsson, S. H. (2022). *Predicting football match outcomes with fantasy league data and deep learning (Doctoral dissertation).*

[16] Lindberg, A., & Söderberg, D. (2020). Comparison of Machine Learning Approaches Applied to Predicting Football Players Performance. *Odr.chalmers.se*. https://hdl.handle.net/20.500.12380/301745

[17] Bangdiwala, M., Choudhari, R., Hegde, A., & Salunke, A. (2022, August 1). Using ML Models to Predict Points in Fantasy Premier League. IEEE Xplore. https://doi.org/10.1109/ASIANCON55314.2022.9909447

[18] Jolliffe, I.T. and Cadima, J. (2016). Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, [online] 374(2065), p.20150202. doi: https://doi.org/10.1098/rsta.2015.0202

[19] Franks, A., Miller, A., Bornn, L., & Goldsberry, K. (2015). Characterizing the spatial structure of defensive skill in professional basketball. *The Annals of Applied Statistics*, *9*(1), 94–121. https://doi.org/10.1214/14-aoas799

[20] Andrienko, G., Andrienko, N., Budziak, G., Dykes, J., Fuchs, G., von Landesberger, T., & Weber, H. (2017). Visual analysis of pressure in football. *Data Mining and Knowledge Discovery*, *31*(6), 1793–1839. https://doi.org/10.1007/s10618-017-0513-2

[21] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

[22] Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, *58*(1), 267–288. https://www.jstor.org/stable/2346178

[23] Chandrashekar, G., & Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, *40*(1), 16–28. https://doi.org/10.1016/j.compeleceng.2013.11.024

[24] Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, *97*(1-2), 273–324. https://doi.org/10.1016/s0004-3702(97)00043-x

[25] Goldsberry, K., & Weiss, E. (2013). The Dwight Effect : A New Ensemble of Interior Defense Analytics for the NBA.

[26] Berrar, D., & Flach, P. (2011). Caveats and pitfalls of ROC analysis in clinical microarray research (and how to avoid them). *Briefings in Bioinformatics*, *13*(1), 83–97. https://doi.org/10.1093/bib/bbr008

[27] Hastie, T., Tibshirani, R., & Friedman, J. (2008). *Springer Series in Statistics The Elements of Statistical Learning Data Mining, Inference, and Prediction Second Edition*. https://hastie.su.domains/Papers/ESLII.pdf

[28] Hutter, F., Lars Kotthoff and Vanschoren, J. (2019). *Automated machine learning : methods, systems, challenges*. Cham Springer.

[29] Lucey, P., Bialkowski, A., Monfort, M., Carr, P.W. and Matthews, I. (2015). 'Quality vs Quantity': Improved Shot Prediction in Soccer using Strategic Features from Spatiotemporal Data.

[30] Kuhn, M., Johnson, K., & Springer Science+Business Media. (2016). *Applied predictive modeling*. Springer.

[31] Leifeld, P., Cranmer, S. J., & Desmarais, B. A. (2018). Temporal Exponential Random Graph Models with btergm: Estimation and Bootstrap Confidence Intervals. *Journal of Statistical Software*, *83*(6). https://doi.org/10.18637/jss.v083.i06

[32] Buuren, S. van, & Groothuis-Oudshoorn, K. (2011). mice: Multivariate Imputation by Chained Equations inR. *Journal of Statistical Software*, *45*(3). https://doi.org/10.18637/jss.v045.i03

[33] Jain, A. K., Duin, P. W., & Jianchang Mao. (2000). Statistical pattern recognition: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(1), 4–37. https://doi.org/10.1109/34.824819

[34] *Time Series - Walk Forward Validation*. (n.d.). Www.tutorialspoint.com. Retrieved July 21, 2023, from https://www.tutorialspoint.com/time_series/time_series_walk_forward_validation.htm#:~:text=In%20time%20series%20modelling%2C%20the

[35] Novack, G. (2020, June 8). *Building a One Hot Encoding Layer with Tensorflow*. Medium. https://towardsdatascience.com/building-a-one-hot-encoding-layer-with-tensorflow-f907d686bf39

[36] Zahangir Alom, Taha, T. M., Yakopcic, C. G., Westberg, S., Paheding Sidike, Mst. Samima Nasrin, Brian Van Essen, Abdul, & Asari, V. K. (2018). *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches.*

[37] Anand, V. (2020, October 30). *vaastav/Fantasy-Premier-League*. GitHub. https://github.com/vaastav/Fantasy-Premier-League

[38] *FIFA22 OFFICIAL DATASET*. (n.d.). Www.kaggle.com. https://www.kaggle.com/datasets/bryanb/fifa-player-stats-database

Appendix A – Python Code

Data Collection

This notebook contains the data collection stage of my project on Predicting EPL Fantasy Football Points. Including some data cleansing required for merging the two data sources I intend to use.

```
#Import the necessary packages
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from fuzzywuzzy import fuzz, process # Required for "fuzzy" matching between
player names
```

FPL Data Import

Here we import the FPL Data set (downloaded from Github)

```
#Import and combine the various seasons we intend to build our model from
def read_csv_files(location, encoding='utf-8'):
    all_dataframes = []
    for file in os.listdir(location):
        if file.endswith('.csv'):
            file_path = os.path.join(location, file)
            df = pd.read_csv(file_path, encoding=encoding)
            file_year = os.path.splitext(file)[0][-4:]  # Get last 4 digits o
f the file name
            df['Year'] = file_year  # Add new column with the Year
            all_dataframes.append(df)

    combined_df = pd.concat(all_dataframes, ignore_index=True)
    return combined_df


folder_location = 'C:/Users/Hazbuk/Documents/University/Project/Data/GameWeek
_Data'
fpl_data = read_csv_files(folder_location)

#Quick visual inspection of the data
fpl_data

'''
Upon inspection of the data from 2021 onwards, "Expected" values were not gat
hered until later seasons.
We would expect these to be valuable in research, however, given how little d
ata is available for them
we shall perform our research without their use.
Fortunately xP (Expected Points) has been gathered for all 3 seasons.
'''
fpl_data[fpl_data['xP'].isna()] # We that there are no null values in 'xP'
```

```
#Basic Information about the FPL Dataset
print("Basic Information:")
print(fpl_data.info())

#Missing values for all columns
print("Missing Values:")
missing_values = fpl_data.isnull().sum()
for column, count in missing_values.items():
    print(f"{column:<30} {count}")

#Data types of columns
print("Data Types:")
for column, dtype in fpl_data.dtypes.items():
    print(f"{column:<30} {dtype}")

#Select numerical columns only
numerical_cols = fpl_data.select_dtypes(include=[np.number]).columns.tolist()

#Correlation matrix (for numerical columns)
print("Correlation Matrix:")
correlation_matrix = fpl_data[numerical_cols].corr()

#Plot the heatmap
plt.figure(figsize=(12, 12))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".1f", annot
_kws={'size': 8})
plt.title("Correlation Matrix", fontsize=15)  # Adjust the title font size
plt.show()
```

FIFA Data Import

```
#Import the FIFA Dataset for combination with FPL Dataset
#This uses the same script previously written to iterate the files
folder_location = 'C:/Users/Hazbuk/Documents/University/Project/Data/FIFA_Dat
a'
fifa_data = read_csv_files(folder_location)

#Quick visual inspection of the data
fifa_data

#Basic Information about the FIFA Dataset
print("Basic Information:")
print(fifa_data.info())

#Missing values for all columns
print("Missing Values:")
missing_values = fifa_data.isnull().sum()
for column, count in missing_values.items():
    print(f"{column:<30} {count}")

#Data types of columns
print("Data Types:")
for column, dtype in fifa_data.dtypes.items():
    print(f"{column:<30} {dtype}")

#Select numerical columns only
numerical_cols = fifa_data.select_dtypes(include=[np.number]).columns.tolist(
)
```

```
#Correlation matrix (for numerical columns)
print("Correlation Matrix:")
correlation_matrix = fifa_data[numerical_cols].corr()

#Plot the heatmap
plt.figure(figsize=(12, 12))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".1f", annot
_kws={'size': 8})
plt.title("Correlation Matrix", fontsize=15)  # Adjust the title font size
plt.show()
```

Merging the Datasets

Now that the data sets are ingested, we need to combine them, from the Literature review we found that Kirstinsson encountered several data cleansing issues while trying to match player names. As such, we need to inspect the various names in each set and see what string manipulations we can employ such that they will match and allow us to join the data successfully.

As part of the process through this step, it has become apparent that to effectively join the data we will need the FIFA data for the corresponding years and not just the latest information on player statistics.

Differences in Club names across the two data sets were noted, these have to be manually altered in order to facilitate the joins.

```
#Firstly we need to replace the club names to match

#Create the mapping to replace the club names
name_mapping = {
    'Brighton & Hove Albion': 'Brighton',
    'West Ham United': 'West Ham',
    'Sheffield United': 'Sheffield Utd',
    'Everton': 'Everton',
    'Fulham': 'Fulham',
    'Wolverhampton Wanderers': 'Wolves',
    'Leeds United': 'Leeds',
    'Leicester City': 'Leicester',
    'Liverpool': 'Liverpool',
    'West Bromwich Albion': 'West Brom',
    'Arsenal': 'Arsenal',
    'Southampton': 'Southampton',
    'Newcastle United': 'Newcastle',
    'Chelsea': 'Chelsea',
    'Crystal Palace': 'Crystal Palace',
    'Tottenham Hotspur': 'Spurs',
    'Manchester United': 'Man Utd',
    'Manchester City': 'Man City',
    'Aston Villa': 'Aston Villa',
    'Burnley': 'Burnley',
    'Watford': 'Watford',
    'Norwich City': 'Norwich',
    'Brentford': 'Brentford',
    'AFC Bournemouth': 'Bournemouth',
    'Nottingham Forest': "Nott'm Forest"
```

```
}

#Replace the occurrences in the fifa data based on the mapping above
fifa_data.replace({"Club": name_mapping}, inplace=True)

#Code used to check for the unique club names in the FIFA data following mapp
ing
#pd.set_option('display.max_rows', 10) #Display more rows for inspection
#fifa_data['Club'].drop_duplicates()

#Code used to check for the unique team names in fpl data
#fpl_data['team'].drop_duplicates()

#Set the team names to filter the fifa dataset to be the same as those in the
fpl dataset
#This proved unuseful as some players were assigned to incorrect teams in the
FIFA data
#Teams = fpl_data['team'].drop_duplicates()
#fifa_data = fifa_data[fifa_data['Club'].isin(Teams)] #filter the fifa data t
o EPL Clubs

#Select only the relevant columns for the joins to inspect
fifa_names = fifa_data[['Name','Club','Year']].drop_duplicates()
#Check the names side by side for players and their respective clubs + the ye
ar
fifa_names

#Used to check the names in detail
for row in fifa_names.values:
    print(row)

#Select only the relevant columns for the joins to inspect
fpl_names = fpl_data[['name','team','Year']].drop_duplicates()
#Check the names side by side for players and their respective clubs + the ye
ar
fpl_names

#Used to check the names in detail
for row in fpl_names.values:
    print(row)

#Let's amend the columns names for the joins to be the same
#This should make the code more readable when joining

print('FIFA columns','\n',fifa_data.columns)
print('FPL columns','\n',fpl_data.columns)

#Rename the columns as required
fifa_data.rename(columns={'Name': 'name','Club': 'team'}, inplace=True)

#We notice that the names in the FIFA Data include some leading and trailing
spaces,
# let us strip these out
fifa_data['name'] = fifa_data['name'].str.strip()

#Reset the index of both DataFrames - required to fix error
fpl_data.reset_index(drop=True, inplace=True)
fifa_data.reset_index(drop=True, inplace=True)
```

```
#Create a Dictionary to store previously found matches
name_matches = {}

#Define a function to find the best match for "name" using fuzzy string match
ing
def find_best_name_match(row, fifa_data):
    index = row.name
    name = row['name']
    team = row['team']
    Year = row['Year']

    #Check if the name was previously matched
    if name in name_matches:
        best_match = name_matches[name]
        #Print included during initial run to ensure correct functionality
        #print(f"Working on row {index}: name={name}, team={team}, Year={Year
}, best_match={best_match} (previously found)")
        return best_match

    #Filter the names based on matching "team" and "Year"
    filtered_names = fifa_data.loc[(fifa_data['team'] == team) & (fifa_data['
Year'] == Year), 'name']

    best_match = None
    max_similarity = 0

    for name_in_list in filtered_names:
        similarity_score = fuzz.token_set_ratio(name, name_in_list)
        if similarity_score > max_similarity:
            max_similarity = similarity_score
            best_match = name_in_list

    #If the maximum similarity score is below a certain threshold, return nul
l
    if max_similarity < 80:  #Set to 80 to avoid over-fitting
        #Print included during initial run to ensure correct functionality
        #print(f"Working on row {index}: name={name}, team={team}, Year={Year
}, best_match=None")
        return np.nan

    #Store the found match in the Dictionary
    name_matches[name] = best_match
    #Print included during initial run to ensure correct functionality
    #print(f"Working on row {index}: name={name}, team={team}, Year={Year}, b
est_match={best_match}")
    return best_match

#Apply the new names to the dataset
fpl_data['processed_name'] = fpl_data.apply(find_best_name_match, fifa_data=f
ifa_data, axis=1)
fifa_data['processed_name'] = fifa_data['name']
```

```
#Check how many names are still missing
print('Current Missing Values: ',len(fpl_data[fpl_data['processed_name'].isna
()]))

#Define a function as above but only using "name" and "team"
def find_best_name_match_unmatched(row, fifa_data, name_matches):
    index = row.name
    name = row['name']
    team = row['team']

    #If the name is already matched in name_matches, use that match
    if name in name_matches:
        #Print included during initial run to ensure correct functionality
        #print(f"Using previously matched name for row {index}: name={name},
team={team}, best_match={name_matches[name]}")
        return name_matches[name]

    #Filter the names from fifa_data based on matching "team" (i.e. excluding
Year)
    filtered_names = fifa_data.loc[(fifa_data['team'] == team), 'name']

    best_match = None
    max_similarity = 0

    for name_in_list in filtered_names:
        similarity_score = fuzz.token_set_ratio(name, name_in_list)
        if similarity_score > max_similarity:
            max_similarity = similarity_score
            best_match = name_in_list

    #If the maximum similarity score is below a certain threshold, return nul
l
    if max_similarity < 80:  #80 again to avoid over-fitting
        #Print included during initial run to ensure correct functionality
        #print(f"Working on unmatched row {index}: name={name}, team={team},
best_match=None")
        return np.nan

    #Store the found match in the Dictionary
    name_matches[name] = best_match
    #Print included during initial run to ensure correct functionality
    #print(f"Working on unmatched row {index}: name={name}, team={team}, best
_match={best_match}")
    return best_match

#Get rows from the joined data where the processed_name is null (unmatched na
mes)
unmatched_data = fpl_data[fpl_data['processed_name'].isnull()]

#Apply the function to find best matches for unmatched names
unmatched_data['processed_name'] = unmatched_data.apply(find_best_name_match_
unmatched, fifa_data=fifa_data, name_matches=name_matches, axis=1)

#Update the processed names in the original fpl_data DataFrame
fpl_data.update(unmatched_data)
```

```
#Check how many names are still missing
print('Current Missing Values: ',len(fpl_data[fpl_data['processed_name'].isna
()]))

#Similar function as above, but specifically where team names match or they a
re NULL
def find_best_name_match_unmatched(row, fifa_data, name_matches):
    index = row.name
    name = row['name']
    team = row['team']

    #If the name is already matched in name_matches, use that match
    if name in name_matches:
        #Print included during initial run to ensure correct functionality
        #print(f"Using previously matched name for row {index}: name={name},
team={team}, best_match={name_matches[name]}")
        return name_matches[name]

    #Filter the names based on matching "team" or where the team is NULL
    filtered_names = fifa_data.loc[(fifa_data['team'] == team) | fifa_data['t
eam'].isnull(), 'name']

    best_match = None
    max_similarity = 0

    for name_in_list in filtered_names:
        similarity_score = fuzz.token_set_ratio(name, name_in_list)
        if similarity_score > max_similarity:
            max_similarity = similarity_score
            best_match = name_in_list

    #If the maximum similarity score is below a certain threshold, return nul
l
    if max_similarity < 80:  #As previously, 80 to avoid over-fitting
        #Print included during initial run to ensure correct functionality
        #print(f"Working on unmatched row {index}: name={name}, team={team},
best_match=None")
        return np.nan

    #Update the dictionary with the new match
    name_matches[name] = best_match
    #Print included during initial run to ensure correct functionality
    #print(f"Working on unmatched row {index}: name={name}, team={team}, best
_match={best_match}")
    return best_match

#Get rows from the joined data where the processed_name is null (unmatched na
mes)
unmatched_data = fpl_data[fpl_data['processed_name'].isnull()]

#Apply the function to find best matches for unmatched names
unmatched_data['processed_name'] = unmatched_data.apply(find_best_name_match_
unmatched, fifa_data=fifa_data, name_matches=name_matches, axis=1)

#Update the processed names in the original fpl_data DataFrame
fpl_data.update(unmatched_data)
```

```
#Check how many names are still missing
print('Total Names: ',len(fpl_data['name'].drop_duplicates()))
print('Names not yet matched: ',len(fpl_data['name'].drop_duplicates())-len(n
ame_matches))

#Create a new approach, as we are missing 290 names following initial fuzzy m
atching

#Extract unique unmatched names from fpl_data
unmatched_names = fpl_data.loc[~fpl_data['name'].isin(name_matches.keys()), '
name'].unique()

#Create a lookup table of best matches for the unmatched names
lookup_table = {}
for i, name in enumerate(unmatched_names):
    #Abbreviate the first name (as the name is often abbreviated in the FIFA
data)
    first_name, *last_name_parts = name.split()
    abbreviated_first_name = first_name[0] if first_name else ''
    abbreviated_name = f"{abbreviated_first_name} {' '.join(last_name_parts)}
"

    best_match_data = process.extractOne(abbreviated_name, fifa_data['name'],
scorer=fuzz.token_set_ratio)
    score = best_match_data[1]
    best_match_name = best_match_data[0]
    if score > 45:  #Lower similarity score as final automated step
        lookup_table[name] = best_match_name

    #Print included during initial run to ensure correct functionality
    #print(f"Working on row {i} of {len(unmatched_names)}: name={name}, best_
match={best_match_name if score > 45 else 'None'}")

#Update the dictionary with new matches
name_matches.update(lookup_table)

#Apply the processed names to the data
fpl_data['processed_name'] = fpl_data['name'].map(name_matches).fillna(fpl_da
ta['name'])

#Check how many names are still missing
print('Current Missing Values: ',len(fpl_data[fpl_data['processed_name'].isna
()]))
```

Following the above steps we now need to do some manual comparisons of the dictionary to correct any errors (of which we clearly have many), this is time consuming but necessary as automation of the process can only take us so far as fuzzy matching is now perfect.

```
#Manual inspection of the name_matches dictionary now that Fuzzy logic is com
plete
name_matches

#Exporting the name list to csv to inspect via Excel for ease
#Convert the Dictionary to a List and then to a DataFrame
items_list = list(name_matches.items())
df = pd.DataFrame(items_list)
```

```
#Export DataFrame to CSV file
output_filename = 'Player_Names.csv'
df.to_csv(output_filename, index=False)
```

Outputting the above and manually inspecting gives us a list of 115 names for which the fuzzy logic has not been successful, we will check these manually and amend the dictionary, once complete we will apply the finished dictionary to the datasets, merge them and we can begin the next stages.

```
#Create a new map to amend the incorrect names which have been identified
def update_name(name1, name2):
    name_map = {
        'Matthew Longstaff': 'M. Longstaff',
        'James Garner': 'J. Garner',
        'Aaron Ramsey': 'A. Ramsey',
        'Stefan Ortega Moreno': 'S. Ortega',
        'Kadan Young': None,
        'Lewis Miley': None,
        'Charlie Robinson': 'C. Robinson',
        'Luke Harris': None,
        'Ahmed El-Sayed Hegazy': 'A. Hegazi',
        'Jonathan Castro Otto': 'Jonny',
        'Johann Berg Gudmundsson': 'J. Guðmundsson',
        'Willian José Da Silva': 'Willian José',
        'Lewis Brunt': None,
        'Lewis Hall': 'L. Hall',
        'Diogo Teixeira da Silva': 'Diogo Jota',
        'Lewis Warrington': 'L. Warrington',
        'Jackson Smith': 'J. Lewis',
        'Rico Lewis': None,
        'Bobby De Cordova-Reid': 'B. Decordova-Reid',
        'Willian Borges da Silva': 'Willian',
        'Diego Da Silva Costa': 'Diego Costa',
        'Alexandre Moreno Lopera': 'Álex Moreno',
        'João Victor Gomes da Silva': None,
        'Hamed Traorè': 'H. Traoré',
        'Ishé Samuels-Smith': None,
        'Francisco Casilla Cortés': None,
        'Jorge Luiz Frello Filho': 'Jorginho',
        'Roberto Jimenez Gago': 'R. Jiménez',
        'Fernando Luiz Rosa': 'Fernandinho',
        'José Ignacio Peleteiro Romallo': 'Jota',
        'Raphael Dias Belloli': 'Raphinha',
        'Karlo Ziger': None,
        'Harry Tyrer': None,
        'Liam Hughes': None,
        'Jaden Philogene-Bidace': None,
        'Tawanda Maswanhise': None,
        'Alex Kral': 'A. Král',
        'Louie Moulden': None,
        'Charlie Whitaker': None,
        'James Storer': None,
        'Filip Marschall': None,
        'Armstrong Oko-Flex': 'A. Okoflex',
        'Jarell Quansah': None,
```

```
        'Kamil Conteh': None,
        'Francisco Jorge Tomás Oliveira': 'Chiquinho',
        'Alejandro Garnacho Ferreyra': 'Alejandro Garnacho',
        'Matthew Craig': None,
        'Tyler Dibling': None,
        'Carlos Ribeiro Dias': 'Cafú',
        'Marcus Oliveira Alencar': 'Marquinhos',
        'Stanley Mills': None,
        'Son Heung-min': 'H. Son',
        'Stefan Bajcetic': None,
        'Matheus Luiz Nunes': 'Matheus Nunes',
        'Kaden Rodney': None,
        'Carlos Henrique Casimiro': 'Casemiro',
        'Lucas Tolentino Coelho de Lima': 'Lucas Paquetá',
        'Renan Augusto Lodi dos Santos': 'Renan Lodi',
        'Harvey Griffiths': None,
        'Nathan Fraser': None,
        'Ben Doak': None,
        'Kristian Sekularac': None,
        'Jimmy Morgan': None,
        'Bashir Humphreys': None,
        'Amario Cozier-Duberry': None,
        'David Ozoh': None,
        'Jack Hinshelwood': None,
        'Gustavo Henrique Furtado Scarpa': 'Gustavo Scarpa',
        'Nathan Butler-Oyedeji': None,
        'Mislav Orsic': 'M. Oršić',
        'Daniel Adu-Adjei': None,
        'Michael Dacosta Gonzalez': None,
        'Dominic Sadi': None,
        'Andrey Nascimento dos Santos': None,
        'Euan Pollock': None,
        'Wanya Marçal-Madivadua': None,
        'Max Kinsey-Wellings': None,
        'Felipe Augusto de Almeida Monteiro': 'Felipe',
        'Jeremiah Chilokoa-Mullen': None,
        'Diogo Pinheiro Monteiro': 'Diogo Monteiro',
        'Oliwier Zych': None,
        'Kaelan Casey': None,
        'George Wickens': None,
        'Romaine Mundle': None,
        'Travis Patterson': None,
        'Cameron Peupion': None,
        'Matthew Dibley-Dias': None,
        'Nico O\'Reilly': None,
        'Shea Charles': None,
        'Kamari Doyle': None,
        'Ethan Wady': None,
        'Samuel Amo-Ameyaw': None,
        'Yago de Santiago Alonso': None
    }
    return name_map.get(name1, name2)

#Update name_matches dictionary based on the above manual entries
for name, incorrect_match in name_matches.items():
    name_matches[name] = update_name(name, incorrect_match)
```

```
#Apply the mapping to the fpl_data
fpl_data['processed_name'] = fpl_data['name'].map(name_matches)

#Perform a quick visual check
fpl_data['processed_name']

#We should now see missing values as per the ones we have removed above only
print('Current Missing Values: ',len(fpl_data[fpl_data['processed_name'].isna
()]))

#Merge on 'processed_name', 'team', and 'Year'
merged_all = pd.merge(fpl_data, fifa_data, on=['processed_name', 'team', 'Yea
r'], how='left')

#Get the remaining rows which were not successfully merged
remaining_fpl = fpl_data.loc[~fpl_data.set_index(['processed_name', 'team', '
Year']).index.isin(merged_all.set_index(['processed_name', 'team', 'Year']).i
ndex)]
remaining_fifa = fifa_data.loc[~fifa_data.set_index(['processed_name', 'team'
, 'Year']).index.isin(merged_all.set_index(['processed_name', 'team', 'Year']
).index)]

#Merge on 'processed_name' and 'Year'
merged_two = pd.merge(remaining_fpl, remaining_fifa, on=['processed_name', 'Y
ear'], how='left')

#Concatenate the results
final_result = pd.concat([merged_all, merged_two])
```

We now need to check the duplicated rows and keep the appropriate rows

```
#Manual inspection of what is duplicated, appears to be one player with two d
ifferent positions
#for simplicity we will just keep the first row for these entries
#pd.set_option('display.max_rows', 80)

#Checking duplicates by selecting only columns relating to the FPL Data
# final_result[final_result[['name_x',
#                  'position',
#                  'team',
#                  'xP',
#                  'assists',
#                  'bonus',
#                  'bps',
#                  'clean_sheets',
#                  'creativity',
#                  'element',
#                  'fixture',
#                  'goals_conceded',
#                  'goals_scored',
#                  'ict_index',
#                  'influence',
#                  'kickoff_time',
#                  'minutes',
#                  'opponent_team',
#                  'own_goals',
#                  'penalties_missed',
```

```
#                  'penalties_saved',
#                  'red_cards',
#                  'round',
#                  'saves',
#                  'selected',
#                  'team_a_score',
#                  'team_h_score',
#                  'threat',
#                  'total_points',
#                  'transfers_balance',
#                  'transfers_in',
#                  'transfers_out',
#                  'value',
#                  'was_home',
#                  'yellow_cards',
#                  'GW',
#                  'Year',
#                  'expected_assists',
#                  'expected_goal_involvements',
#                  'expected_goals',
#                  'expected_goals_conceded',
#                  'starts']].duplicated(keep=False)]

#Filter the dataframe for the duplicates and keep the first entry
final_result = final_result[~final_result.duplicated(keep='first',
                                        subset =
                                        [ 'name_x',
                                          'position',
                                          'team',
                                          'xP',
                                          'assists',
                                          'bonus',
                                          'bps',
                                          'clean_sheets',
                                          'creativity',
                                          'element',
                                          'fixture',
                                          'goals_conceded',
                                          'goals_scored',
                                          'ict_index',
                                          'influence',
                                          'kickoff_time',
                                          'minutes',
                                          'opponent_team',
                                          'own_goals',
                                          'penalties_missed',
                                          'penalties_saved',
                                          'red_cards',
                                          'round',
                                          'saves',
                                          'selected',
                                          'team_a_score',
                                          'team_h_score',
                                          'threat',
                                          'total_points',
                                          'transfers_balance',
```

```
                                        'transfers_in',
                                        'transfers_out',
                                        'value',
                                        'was_home',
                                        'yellow_cards',
                                        'GW',
                                        'Year',
                                        'expected_assists',
                                        'expected_goal_involvements',
                                        'expected_goals',
                                        'expected_goals_conceded',
                                        'starts'])]

#Export the final DataFrame to a CSV file, we can then use this for our futur
e stages
output_filename = 'Combined_FIFA_FPL_Data.csv'
final_result.to_csv(output_filename)
```

Data Processing

This notebook contains the data processing stage of my project on Predicting EPL Fantasy Football Points. Including data conversion/cleaning, handling of missing values, normalisation and feature selection.

```
#Import the necessary packages
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.preprocessing import StandardScaler
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer, KNNImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LassoCV
```

```
#Load the data we exported in the previous step
df = pd.read_csv('Combined_FIFA_FPL_Data.csv', low_memory=False)
```

```
#Check the length of the dataframe
print("\nLength of Frame:", len(df))
```

```
#Let us check for missing values in each of the columns
missing_values = df.isnull().sum().sort_values(ascending=False)
print("\nMissing Values: ")
for index, value in missing_values.items():
    print(f"{index}: {value}")
```

```
#Following the joins, we can drop processed_name from the data
#We can also drop team_y, team_x and Marking - since these are always NULL
#name_y also should be dropped, as this is created as part of the joining pro
cess
df.drop(columns=['processed_name',
                 'team_y',
                 'team_x',
                 'Marking',
                 'name_y'], inplace=True)
'''
We can also see from the above that expected values and starts have not been
collected for many seasons.
As such, it may be worth us trying two approaches, the first, using all histo
rical data we have and
the second using only the seasons where expected values are available. Perhap
s we will see some
significant difference.
'''
#For now, we shall drop those columns also
df.drop(columns=['starts',
                 'expected_assists',
                 'expected_goals',
                 'expected_goals_conceded',
                 'expected_goal_involvements'], inplace=True)
```

```
#There are clearly several extra columns we can remove, namely: Loaned From,
Kit Number, Unnamed: 0,
# ID, Joined, Club Logo, Flag, Photo, Release Clause, Contract Valid Until
#These all should have no value in future points prediction
df.drop(columns=['Loaned From',
                 'Kit Number',
                 'Unnamed: 0',
                 'ID',
                 'Joined',
                 'Club Logo',
                 'Flag',
                 'Photo',
                 'Release Clause',
                 'Contract Valid Until'], inplace=True)

#Select the categorical columns for inspection
df_categorical = df.select_dtypes(include=[object])
df_categorical

#From inspection we can see several columns that can be remove, Nationality s
hould have little or no impact
# or at the very least will be explained by other variables. Body Type, Real
Face, Position,
# and Best Position can also all be removed
df.drop(columns=['Nationality',
                 'Body Type',
                 'Real Face',
                 'Position',
                 'Best Position'], inplace=True)

#Additionally kickoff_time should be a datetime element, however, we can also
exclude this from our work
# for simplicity
df.drop(columns=['kickoff_time'], inplace=True)

#We also notice that Value, Wage, Height and Weight need some conversion to a
ccurately represent the
# numerical variables that they are.

#Let's start with Height
#Function to convert height in feet and inches to centimeters
def convert_to_cm(height):
    if pd.notna(height) and "'" in height:  # If height is not NULL and in fe
et and inches format
        feet, inches = map(int, height.split("'"))
        total_inches = (feet * 12) + inches
        return total_inches * 2.54
    elif pd.notna(height) and "cm" in height:  # If height is not NULL and in
centimeters format
        return int(height.replace("cm", ""))
    else:
        return None

#Apply the function to the 'height' column
df['Height'] = df['Height'].apply(convert_to_cm)
```

```
#Weight
#Function to convert weight in pounds to kilograms
def convert_to_kg(weight):
    if pd.notna(weight) and "lbs" in weight:  # If weight is not NULL and in
pounds format
        pounds = int(weight.replace("lbs", ""))
        return pounds * 0.453592
    elif pd.notna(weight) and "kg" in weight:  # If weight is not NULL and in
kilograms format
        return int(weight.replace("kg", ""))
    else:
        return None

#Apply the function to the 'weight' column
df['Weight'] = df['Weight'].apply(convert_to_kg)

#Value and Wage
#Function to convert values to numeric
def convert_value_to_numeric(value):
    if pd.notna(value):
        value_lower = value.lower() # Remove case-sensitivity
        multiplier = 1
        if 'm' in value_lower:
            multiplier = 1e6 # m denoting million
        elif 'k' in value_lower:
            multiplier = 1e3 # k denotion thousand

        #Extract numeric part from the value string
        numeric_value = ''.join(filter(str.isdigit, value))

        return float(numeric_value) * multiplier
    else:
        return None

#Apply the function to the 'value' column
df['Value'] = df['Value'].apply(convert_value_to_numeric)

#Apply the same function to the 'wage' column
df['Wage'] = df['Wage'].apply(convert_value_to_numeric)

#Finally for these early stages, since our data is TimeSeries in nature, we s
hould sort the data as such.
#We can do this by sorted on the GW (Game Week) and Year columns

# Sort the DataFrame on "Year" and "GW" columns in ascending order
df = df.sort_values(by=['Year', 'GW'], ascending=True)

#To handle missing values we need to split into numerical and categorical dat
a
df_numeric = df.select_dtypes(include=[np.number])
df_categorical = df.select_dtypes(exclude=[np.number])

#As discussed in my Methodology there are several methods of handling missing
values
#Method 1: Deletion - deleting rows with missing values
df_deletion = df.dropna()
```

```
#Method 2: Mean Imputation - replacing missing values with the mean of the co
lumn
df_mean_imputation = df_numeric.fillna(df_numeric.mean())

#Method 3: Multivariate Imputation - replacing missing values by
# modeling each feature with missing values as a function of other features
imputer = IterativeImputer(max_iter=10, random_state=0)
df_imputed = df_numeric.copy()
df_imputed[df_numeric.columns] = imputer.fit_transform(df_numeric)

#Method 4: KNN Imputation - replacing missing values using the k-Nearest Neig
hbours approach
knn_imputer = KNNImputer(n_neighbors=5)  #Use 5 nearest neighbours
df_knn_imputed = df_numeric.copy()
df_knn_imputed[df_numeric.columns] = knn_imputer.fit_transform(df_numeric)
```

I expect KNN to be a good fit for imputing missing values, largely because there is considerable spread across the data, such that just choosing the global mean may cause some problems. Unfortunately it isn't perfect as we have some strong correlalations between some variables, as shown below. Particularly in relation to the FIFA Data.

```
#Correlation matrix (for numerical columns)
print("Correlation Matrix:")
correlation_matrix = df_numeric.corr()

#Plot the heatmap
plt.figure(figsize=(12, 12))
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm')
plt.title("Correlation Matrix", fontsize=15)  # Adjust the title font size
plt.show()

#In the interest of being able to try each method as efficiently as possible
#we set below and can re-run our code as necessary:
# df_numeric = df_deletion
# df_numeric = df_mean_imputation
# df_numeric = df_imputed
df_numeric = df_knn_imputed

#Finally concatenate the numerical and categorical dataframes together again
df = pd.concat([df_numeric, df_categorical], axis=1)

#As discussed in my Methodology, there are 3 methods we will use to identify
outliers
# these are detailed below

#Select only numeric columns for Outlier detection
numeric_cols = df.select_dtypes(include=[np.number])

#Identify outliers using box plots -
# this proved too messy, we will recheck following dimensionality reduction
# for column in df.columns:
#     if df[column].dtype in ['int64', 'float64']:  #Numerical only
#         plt.figure(figsize=(12, 6))
#         df.boxplot([column])
#         plt.title(f'Boxplot of {column}')
#         plt.show()
```

```
#Identify outliers using z-score
z_scores = stats.zscore(df.select_dtypes(include=[np.number]))  # Apply z-sco
re for numerical columns only
df_z_scores = pd.DataFrame(z_scores, columns=df.select_dtypes(include=[np.num
ber]).columns)
outliers_z_scores = (df_z_scores > 3).sum().sort_values(ascending=False)  # C
onsider values with z-score > 3 as outliers

print("\nOutliers identified using Z-Scores: ")
print(outliers_z_scores)

#Identify outliers using IQR
Q1 = numeric_cols.quantile(0.25)
Q3 = numeric_cols.quantile(0.75)
IQR = Q3 - Q1
print("\nIQR Values: ")
print(IQR)

#Identify the outliers
outliers = ((numeric_cols < (Q1 - 1.5 * IQR)) | (numeric_cols > (Q3 + 1.5 * I
QR)))

#Print the outliers
print("\nRows containing at least one \'outlier\' in the DataFrame:")
print(len(outliers.loc[outliers.any(axis=1)]))
```

Clearly we can see considerable “outliers”, but this is to be expected given the nature of the data. We will perform some standardisation and then revisit these outliers again later after we encode our categorical variables and perform dimensionality reduction.

```
#Normalise data using standard scaler
scaler = StandardScaler()
df_normalized = pd.DataFrame(scaler.fit_transform(df.select_dtypes(include=[n
p.number])),
                             columns=df.select_dtypes(include=[np.number]).co
lumns)
```

Now that we’ve completed our normalisation, let’s look at our categorical variables, ideally we should use One-Hot encoding for those which are nominal and Label encoding for those which are ordinal.

```
#Select the categorical columns for inspection
df_categorical = df.select_dtypes(include=[object])
df_categorical

#We can see from inspection that the categorical variables are all nominal (i
.e. they have no order)
#As such, One-Hot encoding is appropriate for all our variables.

#Now is a good time to convert name_x into some ID (since it avoids our norma
lisation)
#We will do this using factorize()
df_categorical['ID'] = pd.factorize(df['name_x'])[0]

#In order to let us convert these back from our final models, we will create
a dictionary also
```

```
id_name_dict = dict(zip(df_categorical['ID'], df_categorical['name_x']))
#We can now also drop the name_x column
df_categorical.drop(columns=['name_x'], inplace=True)

#Perform one-hot encoding on the categorical columns
one_hot_encoded = pd.get_dummies(df_categorical)

#Combine the one-hot encoded and normalised dataframes
df_processed = pd.concat([df_normalized, one_hot_encoded], axis=1)
```

Random Forest Regressor (Feature Selection)

We are now ready to perform our dimensionality reduction. Random Forest Regressors can handle both numerical and categorical data as well as not requiring data to be scaled. They also provide a very easy to interpret measure of feature importance, as such, we'll use this first and re-visit if needed.

```
#Since we are looking to predict'total_points' let's split these into X and y
X = df_processed.drop('total_points', axis=1)
y = df_processed['total_points']

#Since BPS and Bonus are directly related to our output total_points, we shal
l remove them
# (they are essentially sub sets of this value)
X.drop(columns=['bps',
                'bonus'], inplace=True)

#Splitting into training and test datasets (80-20 split seems standard from m
y literature review)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand
om_state=0)

#Select the RandomForestRegressor
model = RandomForestRegressor(n_estimators=100, random_state=0)
model.fit(X_train, y_train)

#Get feature importances
importances = model.feature_importances_
f_importances = pd.Series(importances, X.columns) # Feature importance alongs
ide the Variable
f_importances.sort_values(ascending=False, inplace=True) # Sorting

#Print the importances for visual inspection
print(f_importances)

#Select only the features with importance above a certain threshold
selected_features = f_importances[f_importances > 0.005].index

print(selected_features)

# Refit the model using only the selected features
model.fit(X_train[selected_features], y_train)

#Plotting feature importances
plt.figure(figsize=(10,8))
f_importances[selected_features].sort_values().plot(kind='barh')
plt.title('Feature Importances')
```

```
plt.xlabel('Importance')
plt.show()
```

Lasso regularization (Feature Selection)

Since all variables have been scaled and one-hot encoding has been used on categorical variables, we could consider using Lasso regularisation for feature selection. Lasso (Least Absolute Shrinkage and Selection Operator) has the ability to shrink the coefficients of less important features to exactly 0, effectively performing feature selection.

```
#Define the model
lasso = LassoCV(cv=5)

#Fit the model
lasso.fit(X_train, y_train)

#Convert coefficients to a pandas series
lasso_coeffs = pd.Series(lasso.coef_, index = X_train.columns)

print(lasso_coeffs)

#Select features which coefficients are not 0
selected_features = lasso_coeffs[lasso_coeffs!=0].index

print(selected_features)

#Plotting feature importances
plt.figure(figsize=(10,8))
lasso_coeffs[selected_features].sort_values().plot(kind='barh')
plt.title('Feature Importances')
plt.xlabel('Coefficient Value')
plt.show()
```

Lasso appears to provide us with the best selection of Features based, particularly when taking into consideration features that appeared relevant in other works explored during my Literature Review. As such, we will use this selection of Features for our final models.

```
df_processed

#Assign the final output to a new DataFrame
df_ML = df_processed[['total_points',
                      'xP',
                      'assists',
                      'clean_sheets',
                      'goals_scored',
                      'influence',
                      'minutes',
                      'own_goals',
                      'penalties_saved',
                      'red_cards',
                      'saves',
                      'yellow_cards',
                      'ID']]

#Export the final DataFrame to a CSV file, we can then use this for our futur
e stages
output_filename = 'Machine_Learning_Data.csv'
df_ML.to_csv(output_filename)
```

```
#Convert the Dictionary to a List and then to a DataFrame
items_list = list(id_name_dict.items())
id_name_df = pd.DataFrame(items_list)

#Export to CSV file
output_filename = 'id_name_list.csv'
id_name_df.to_csv(output_filename, index=False)
```

Test & Train

This notebook contains the Test & Train stage of my project on Predicting EPL Fantasy Football Points. Including Linear Regression modelling, Random Forest modelling and XGBoost modelling. As well as interpretation of the models efficacy based on their R-squared, MAE and RMSE values including plots of Actual vs Predicted values.

```
#Import the necessary packages
import pandas as pd
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from tensorflow import keras
from tensorflow.keras.layers import LSTM, Dense

#Load the data we exported in the previous step
df = pd.read_csv('Machine_Learning_Data.csv')
df.drop(columns='Unnamed: 0', inplace=True)

#Load the ID & Name list
id_name_list = pd.read_csv('id_name_list.csv')
```

Linear Regression

Linear Regression is the most simple model and will be used as a baseline for comparison against the other models

```
#Define features and target
X = df.drop(['total_points', 'ID'], axis=1)
y = df['total_points']

#Split the data into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2, # 80-20 sp
lit as per Literature Review
                                                    shuffle=False) # No shuff
le due to time-series nature

#Create a Linear Regression model
model = LinearRegression()

#Train the model
model.fit(X_train, y_train)

#Use the model to make predictions
y_pred = model.predict(X_test)

#Print Mean Absolute Error, Root Mean Squared Error and R-Squared values
```

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,
y_pred)))
print('R-Squared:', metrics.r2_score(y_test, y_pred))

#Plot actual vs predicted values
plt.scatter(y_test, y_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', l
w=2)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Total Points')
plt.show()

#Checking the ID for Kevin De Bruyne
#We will use him to check the outputs from our models
id_name_list[id_name_list['1']=='Kevin De Bruyne']

#Function to predict total_points given an ID
def predict_total_points(ID):
    features = df[df['ID'] == ID].drop(['total_points', 'ID'], axis=1)
    prediction = model.predict(features)
    return prediction

#Test the point prediction
ID = input("Enter an ID: ") #
print(f"The predicted total points for ID {ID} is: {sum(predict_total_points(
int(ID)))}")
```

We can see some heteroscedasticity, this means our models performance is varying across different ranges, perhaps we are not capturing some non-linear relationship. However when we test the total points for a strong player like KDB we can see it is only 11 points off the actual for that season (183), so anecdotally, we appear to have a reasonable model even at this early stage.

The linear regression model seems to perform quite well with a high R-squared value of 90.24%, indicating a good fit to the data. The MAE (0.15) and RMSE (0.29) values provide a sense of the average error magnitude, with the RMSE giving more weight to larger errors. Given that most weeks players score anywhere between -4 and 23 points (in this season) that average error is quite reasonable.

Random Forests

These are our next level of complexity, still relatively easy to understand and interpret

```
#Features and target variable
X = df.drop(['total_points', 'ID'], axis=1)
y = df['total_points']

#Hyperparameters for Grid Search
params = {
    'n_estimators': [50, 100, 200],
    'max_depth': range(3, 10),
    'min_samples_split': [2, 5, 10]
}

#Initialise Random Forest and TimeSeriesSplit
```

```
model = RandomForestRegressor(random_state=42)
tscv = TimeSeriesSplit(n_splits=3) # Representing 3 seasons of data gathered

#Store the best models from each split
best_models = []

for i, (train_index, test_index) in enumerate(tscv.split(X)):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    #Grid Search for hyperparameter tuning
    grid_search = GridSearchCV(estimator=model, param_grid=params, cv=3, n_jo
bs=-1)
    grid_search.fit(X_train, y_train)
    best_models.append(grid_search.best_estimator_)

    #Predictions and Evaluation for the current split
    y_pred = grid_search.best_estimator_.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)

    print(f'Random Forest: Performance for split {i+1}:')
    print(f'MAE: {mae}')
    print(f'RMSE: {rmse}')
    print(f'R-Squared: {r2}')

    plt.figure(figsize=(10, 5))
    plt.scatter(y_test, y_pred)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--
', lw=2)
    plt.xlabel('Actual Total Points')
    plt.ylabel('Predicted Total Points')
    plt.title(f'Random Forest: Actual vs Predicted Total Points for split {i+
1}')
    plt.show()

#Function to predict points based on a chosen model
def predict_total_points(ID, chosen_model):
    features = df[df['ID'] == ID].drop(['total_points', 'ID'], axis=1)
    prediction = chosen_model.predict(features)
    return prediction

#Test the point prediction
ID = input("Enter an ID: ")
chosen_model = best_models[2] # Using the last model as an example.
print(f"The predicted total points for ID {ID} is: {sum(predict_total_points(
int(ID), chosen_model))}")
```

We still see some heteroscedasticity in our models even when splitting across time sections, this means our models performance is varying across different ranges, it is possible there are some other sub groups which should be captured in future modelling to provide better results. When we test the total points for a strong player like KDB we can see it is further out, now almost 21 points off the actual for that season (183).

The Random Forest model appears to perform very well, capturing over 93% of the variability in the target variable. The error metrics, both MAE (0.094) and RMSE (0.249), are quite low, suggesting that on average, the model's predictions are close to the actual observations.

Gradient Boosting Machine Learning: XGBoost

Another step further into complexity, as per my Methodology this is the next step in our process before we finish with Deep Learning Models.

```
#Define features and target
X = df.drop(['total_points', 'ID'], axis=1)
y = df['total_points']

#Initialise the model and define the parameters
model = XGBRegressor(random_state=42)
params = {
    'n_estimators': [50, 100, 200],
    'max_depth': range(3, 10),
    'learning_rate': [0.01, 0.1, 0.2]
}

#TimeSeriesSplit for walk-forward validation
tscv = TimeSeriesSplit(n_splits=3)  # Representing 3 seasons of data gathered

#Store the best models from each split
best_models = []

for i, (train_index, test_index) in enumerate(tscv.split(X)):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    #Grid search to find optimal parameters
    grid_search = GridSearchCV(estimator=model, param_grid=params, cv=5, n_jo
bs=-1)
    grid_search.fit(X_train, y_train)
    best_model = grid_search.best_estimator_

    #Store the best model
    best_models.append(best_model)

    #Model Evaluation
    y_pred = best_model.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)

    print(f'XGBoost Performance for split {i+1}:')
    print(f'MAE: {mae}')
    print(f'RMSE: {rmse}')
    print(f'R-Squared: {r2}')

    #Plotting real vs predicted values
    plt.figure(figsize=(10, 5))
    plt.scatter(y_test, y_pred)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--
```

```
', lw=2)
    plt.xlabel('Actual')
    plt.ylabel('Predicted')
    plt.title(f'XGBoost: Actual vs Predicted for split {i+1}')
    plt.show()

#Test the point prediction
ID = input("Enter an ID: ")
chosen_model = best_models[2] # Using the last model as an example.
print(f"The predicted total points for ID {ID} is: {sum(predict_total_points(
int(ID), chosen_model))}")
```

Again, we see some heteroscedasticity in our models even when splitting across time sections, this means our models performance is varying across different ranges. As before, it seems likely there are some other sub groups which should be captured in future modelling to provide better results. Given XGBoost is not strict with its assumptions like Linear Regression, we would expect the problem is not caused by any non-linear relationship not being captured.

When we test the total points for a strong player like KDB we can see it is slightly improved, now 19 points off the actual for that season (183).

The XGBoost model seems to perform well; it captures nearly 93.58% of the variability in the target variable, and both error metrics, MAE (0.092) and RMSE (0.240), are relatively low. This suggests that, on average, the model's predictions are quite close to the actual values.

Deep Learning

This notebook contains the Deep Learning stage of my project on Predicting EPL Fantasy Football Points. Consisting only of LSTM modelling. As well as interpretation of the model's efficacy based on its R-squared, MAE and RMSE values including plots of Actual vs Predicted values.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
from keras.optimizers import Adam

#Load the data we exported in the previous step
df = pd.read_csv('Machine_Learning_Data.csv')
df.drop(columns='Unnamed: 0', inplace=True)

#Load the ID & Name list
id_name_list = pd.read_csv('id_name_list.csv')

#Checking the ID for Kevin De Bruyne
#We will use him to check the outputs from our models
id_name_list[id_name_list['1']=='Kevin De Bruyne']

#Split the dataset into train and test sets (80-20 split)
train_size = int(len(df) * 0.8)
train, test = df.iloc[:train_size], df.iloc[train_size:]

#Split the data into features and target for both train and test
X_train = train.drop(['total_points', 'ID'], axis=1).values
y_train = train['total_points'].values
X_test = test.drop(['total_points', 'ID'], axis=1).values
y_test = test['total_points'].values

#Reshape the data for LSTM (samples, time steps, features)
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

#Create the LSTM model
model = Sequential()
model.add(LSTM(units=256, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.5))
model.add(Dense(1))
model.compile(optimizer=Adam(learning_rate=0.0004), loss='mean_squared_error'
)

#Train the model
history = model.fit(X_train, y_train, epochs=250, batch_size=33, validation_d
ata=(X_test, y_test), verbose=1, shuffle=False)

#Predictions
y_pred = model.predict(X_test)

#Performance metrics
```

```
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f'MAE: {mae}')
print(f'RMSE: {rmse}')
print(f'R-Squared: {r2}')

#Plot actual vs predicted values
plt.figure(figsize=(10, 5))
plt.scatter(y_test, y_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', l
w=2)
plt.xlabel('Actual Total Points')
plt.ylabel('Predicted Total Points')
plt.title(f'LSTM: Actual vs Predicted Total Points')
plt.show()



#Plot actual vs predicted values
plt.figure(figsize=(10, 5))
plt.scatter(y_test, y_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', l
w=2)
plt.xlabel('Actual Total Points')
plt.ylabel('Predicted Total Points')
plt.title(f'LSTM: Actual vs Predicted Total Points')
plt.show()

#Function to predict 'total_points' given an ID
def predict_total_points(ID):
    features = df[df['ID'] == ID].drop(['total_points', 'ID'], axis=1).values
    features = np.reshape(features, (features.shape[0], 1, features.shape[1])
)
    prediction = model.predict(features)
    return prediction

#Test the point prediction
ID = input("Enter an ID: ")
print(f"The predicted total points for ID {ID} is: {sum(predict_total_points(
int(ID)))}")
```

For our final model we see still some heteroscedasticity. It is now almost certain there are some other sub groups or clusters which should be captured in future modelling to provide better results.

When we test the total points for a strong player like KDB we can see it is slightly improved again, now close to 16 points off the actual for that season (183).

The LSTM deep learning model appears to perform well based on our metrics. It accounts for about 93.04% of the variability in the target variable, and the error metrics, MAE (0.109) and RMSE (0.248), suggest relatively close predictions to the actual values, on average.

However, LSTMs and deep learning models, in general, have high capacity and can easily overfit to training data. As such, we should take these results with caution.

Appendix B – Dataset Locations

FPL Data available at: https://github.com/vaastav/Fantasy-Premier-League/tree/master

FIFA Data available at: https://www.kaggle.com/datasets/bryanb/fifa-player-stats-database

Appendix C – Certificate of Ethics Approval

Certificate of Ethical Approval

Applicant:	Harris Boyle
Project Title:	Using Machine Learning Methods to Predict EPL Fantasy Football Points

This is to certify that the above named applicant has completed the Coventry University Ethical Approval process and their project has been confirmed and approved as Low Risk

Date of approval:	03 Jun 2023
Project Reference Number:	P154629