

**FANTASY PREMIER LEAGUE
TEAM OPTIMAZATION USING
GUIDED LEARNING MODELS AND SOCIAL MEDIA SENTIMENT
ANALYSIS**

Submitted in partial fulfillment of the requirements for the degree of

**Bachelor of Technology
in
Computer Science and Engineering**

by

Rajarshi Saha (20BCT0163)

Ankan Ray (20BCE2304)

Under the guidance of

Dr.Ebenezer Juliet

Associate Professor Senior

SCOPE, VIT, Vellore



May, 2024

ACKNOWLEDGEMENTS

First and foremost, I extend my heartfelt thanks to Dr. Ebenezer Juliet for her invaluable guidance, support, and encouragement throughout this journey. Her expertise, patience, and unwavering commitment have been instrumental in shaping this project and my growth as a scholar.

I am deeply thankful to my teammates and collaborators whose dedication and hard work have enriched this project with diverse perspectives and insights. Your collaborative spirit and shared vision have made this endeavour both rewarding and fulfilling.

I also extend my appreciation to the faculty members and experts who generously shared their knowledge, feedback, and resources, enriching the quality of this work.

Last but not least, I would like to express my gratitude to VIT, Vellore for providing me with the opportunity, resources, and platform to pursue this capstone project and for fostering an environment conducive to intellectual growth and exploration.

Thank you to all who have contributed to this project in ways big and small. Your support has been invaluable, and I am truly grateful for the opportunity to undertake this endeavour.

Ankan Ray
Rajarshi Saha

DECLARATION

I hereby declare that the thesis entitled "Fantasy Premier League Team Optimazation Using Guided Learning Models And Social Media Sentiment Analysis " submitted by us, for the award of the degree of Bachelor of Technology in Computer Science Engineering to VIT is a record of bonafide work carried out by me under the supervision of *Dr.Ebenezer Juliet*.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date : 8th May. 2024


Signature of the Candidate

DECLARATION

I hereby declare that the thesis entitled "Fantasy Premier League Team Optimazation Using Guided Learning Models And Social Media Sentiment Analysis" submitted by us, for the award of the degree of Bachelor of Technology in Computer Science Engineering to VIT is a record of bonafide work carried out by me under the supervision of *Dr. Ebenezer Juliet*.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date : 02 May 2024

Ragavshi Saha
Signature of the Candidate

CERTIFICATE

This is to certify that the thesis entitled "Fantasy Premier League Team Optimazation Using Guided Learning Models And Social Media Sentiment Analysis" submitted by **Rajarshi Saha 20BCE2304, SCOPE, VIT** and **Ankan Ray 20BCE2304, SCOPE, VIT**, for the award of the degree of *Bachelor of Technology in Computer Science Engineering*, is a record of bonafide work carried out by him / her under my supervision during the period, 01. 12. 2018 to 30.04.2019, as per the VIT code of academic and research ethics.

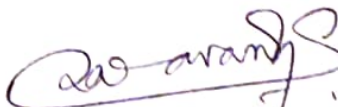
The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 08 May 2024


Signature of the Guide


Internal Examiner


External Examiner

Head of the Department
Computer Science and
Engineering

Head of the Department
Computer Science and Engineering
with Specialization in IOT

CERTIFICATE

This is to certify that the thesis entitled “Fantasy Premier League Team Optimazation Using Guided Learning Models And Social Media Sentiment Analysis ” submitted by **Rajarshi Saha 20BCE2304, SCOPE, VIT** and **Ankan Ray 20BCE2304, SCOPE, VIT**, for the award of the degree of *Bachelor of Technology in Computer Science Engineering*, is a record of bonafide work carried out by him / her under my supervision during the period, 01. 12. 2018 to 30.04.2019, as per the VIT code of academic and research ethics.

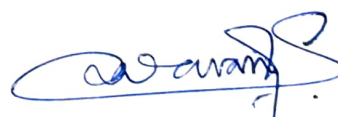
The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 8th May, 2024


Signature of the Guide


Internal Examiner


External Examiner

Head of the Department
Computer Science and
Engineering

Head of the Department
Computer Science and Engineering
with Specialization in IOT

Executive Summary

Our project focuses on optimizing Fantasy Premier League (FPL) team management and strategy across 38 game weeks. Leveraging past data from 3 seasons and current season data, we predict player points using a feature-rich model encompassing player value, position, recent game stats, opponent analysis, and more. We employ a Light GBM model, enhanced with SARIMAX-predicted game scores, for precise predictions.

Dynamic Programming guides squad selection, transfers, and game chip usage, ensuring adherence to FPL rules. A React-based frontend carousel showcases recommended teams, starting XI, bench, predicted points, captains, transfers, and game chips, providing managers with a comprehensive view for informed decision-making.

Our approach integrates advanced analytics, machine learning, and frontend development, offering FPL managers a strategic edge and immersive experience throughout the season.

CONTENTS		Page No.
	Acknowledgement	i.
	Executive Summary	ii.
	Table of Contents	iii.
	List of Figures	iv.
	List of Tables	v.
1	INTRODUCTION	1
	1.1 Objectives	1
	1.2 Motivation	1
	1.3 Background	1
2	PROJECT DESCRIPTION AND GOALS	2
	2.1 Survey on Existing System	3
	2.2 Research Gap	13
	2.2 Problem Statement	14
3	TECHNICAL SPECIFICATION	14
	3.1 Requirements	
	3.1.1 Functional	14
	3.1.2 Non-Functional	
	3.2 Feasibility Study	
	3.2.1 Technical Feasibility	16
	3.2.2 Economic Feasibility	16
	3.2.3 Social Feasibility	17
	3.3 System Specification	
	3.3.1 Hardware Specification	17
	3.3.2 Software Specification	18
4	DESIGN APPROACH AND DETAILS	19
	4.1 System Architecture	19
	4.2 Design	21
	4.2.1 Data Flow Diagram	21
	4.2.2 Use Case Diagram	21
	4.2.3 Class Diagram	22

	4.2.4 Sequence Diagram	22
	4.3 Constraints, Alternatives and Tradeoffs	23
5	SCHEDULE, TASKS AND MILESTONES	24
	5.1 Gantt Chart	24
	5.2 Module Description	
	5.2.1 Data Scrapping	24
	5.2.2 Data Cleaning	24
	5.2.3 Data Assimilation	26
	5.2.4 Data Engineering	26
	5.2.5 Model Training	28
	5.2.6 Squad Optimization	30
	5.2.7 Sentiment Analysis	35
	5.2.8 Frontend	36
	5.3 Testing	
	5.3.1 Unit Testing	37
	5.3.2 Integration Testing	39
6	PROJECT DEMONSTRATION	41
7	RESULT & DISCUSSION	44
8	SUMMARY	48
9	REFERENCES	50
	APPENDIX A – SAMPLE CODE	51

List of Figures

Figure No.	Title	Page No.
1.0	System Architecture	19
1.1	System Overview	20
1.2	Dataflow	21
1.3	Use Case	21
1.4	Class	22
1.5	Sequence	22
1.6	Gantt Chart	24
2.1	SariMax	28
2.2	Objective Function	31
2.3	Constraints 1.0	32
2.4	Constraints 2.0	35
3.1	Player Details	42
3.2	Sample Frontend 1	42
3.3	Sample Frontend 2	42
3.4	Sample Frontend 3	43
3.5	Predicted points comparison	47
3.6	FPL points comparison against other Managers	48

List of Tables

Table No.	Title	Page No.
T1	Literature Survey	3

1. INTRODUCTION

1.1 OBJECTIVE

- Develop a comprehensive platform that utilizes data engineering, machine learning, and artificial intelligence to analyze and predict player performance and points for each game week based on historical data and relevant factors.
- Assist FPL managers in making informed decisions regarding player selection, transfers, and team formation by providing actionable insights and recommendations tailored to individual team compositions and preferences.
- Prioritize fixtures and plan strategies for maximizing points while adhering to budget constraints, optimizing team selection and transfers for optimal performance.
- Enhance user experience and engagement by providing intuitive interfaces and personalized recommendations that improve usability and satisfaction.
- Continually improve and refine the platform through feedback and iteration, ensuring its relevance and effectiveness for FPL managers, thus contributing to ongoing success and satisfaction within the FPL community.

1.2 MOTIVATION

The aim of the Fantasy Premier League (FPL) Assistant project is to revolutionize team management for FPL managers by leveraging data analytics, machine learning, and artificial intelligence. By addressing the challenges faced by FPL managers in player selection, fixture planning, and budget management, the aim is to redefine the FPL experience and empower managers to maximize their points throughout the season.

1.3 BACKGROUND

In the realm of Fantasy Premier League (FPL), the blend of data analytics and strategic decision-making is pivotal for success. Recognizing the challenges FPL managers face in player selection, fixture planning, and maximizing points within budget constraints, our platform aims to revolutionize team management.

By leveraging data engineering, machine learning, and artificial intelligence, our platform provides FPL managers with invaluable insights and predictive analytics.

Our model meticulously plans for each game week, prioritizing fixtures based on team form, fixture difficulty ratings, and player fatigue.

We calculate predicted team scoring and conceding indices, enabling precise player point projections. For attackers and midfielders, factors such as team scoring indices, match fitness, and player form are considered. Defenders and goalkeepers are evaluated based on predicted team conceding indices and save performance.

Additionally, our platform incorporates a player popularity model, utilizing data from FPL Analytics and Twitter sentiment analysis to gauge player popularity and sentiment within the FPL community.

To optimize team selection, a statistical model predicts the best permutations of the playing XI and bench players, maximizing points while adhering to budget constraints. Time-series modelling ensures accurate predictions of player points over successive game weeks.

Through the integration of advanced analytics and intuitive interfaces, our platform aims to empower FPL managers, redefining the FPL experience and enabling managers to navigate the complexities of the Premier League with confidence and success.

2. PROJECT DESCRIPTION AND GOALS

2.1 SURVEY ON EXISTING SYSTEM

T1 Literature Survey

1. Player Recommendation System for Fantasy Premier League using Machine Learning	Vimal Rajesh, P Arjun, Kunal Ravikumar Jagtap, Suneera C M 2022 19th International Joint Conference on Computer Science and Software Engineering (JCSSE)	<p>The paper aims to enhance decision-making for Fantasy Premier League (FPL) players by providing data-driven recommendations.</p> <p>FPL is a popular online game where participants create virtual teams of real-life football players and earn points based on their performance in actual matches.</p> <p>The authors recognize the favoritism bias, where players tend to select players from their favorite teams rather than making informed choices.</p> <p>They propose a recommendation system that leverages machine learning to suggest optimal player combinations.</p> <p>The system uses data extracted from the FPL API.</p> <p>The testing period corresponds to the English Premier League 2021–22 season.</p> <p>The paper takes into consideration the Form, Return on Investment, Fixture Difficulty Rating, Bonus Points System, Points Per Game, and Influence Creativity Threat to determine the points of a player.</p> <p>The paper proposes Random Forest and Gradient Boosting Machines to generate expected points for a player.</p> <p>The paper uses Mean Absolute Error (MAE) is used as a metric to evaluate the model.</p> <p>It explores statistical analysis and data science techniques to generate better recommendations.</p>
---	---	---

<p>2.</p> <p>Sports Analytics algorithms for performance prediction</p>	<p>Konstantinos Apostolou, Christos Tjortjis</p> <p>2019, 10th International Conference on Information, Intelligence, Systems and Applications (IISA)</p>	<p>The dissertation is divided into two major parts. The first part comprises a literature review of existing technologies related to sports analytics.</p> <p>The second part focuses on experiments conducted primarily using football data.</p> <p>The first step involves gathering relevant data. In the context of football, this data could include player statistics, match results, team formations, and more.</p> <p>Once collected, the data needs preprocessing. This step includes handling missing values, normalizing features, and ensuring data consistency.</p> <p>Feature engineering is crucial for building effective predictive models. It involves creating new features or transforming existing ones to improve model performance.</p> <p>For player performance prediction, features might include player position, playing time, goals scored, assists, passes completed, and defensive actions.</p> <p>The model tries the following algorithms Naïve Bayes, Decision Tree, Random Forest, KNN, SVM (rbf kernel), SVM (poly kernel), XGBoost</p> <p>The selected algorithm(s) are trained on the preprocessed data. The model learns patterns from historical data.</p> <p>Evaluation metrics (e.g., accuracy, precision, recall) assess how well the model performs.</p> <p>Once the model is trained, it can predict player performance based on input features.</p> <p>Insights gained from predictions can guide team management decisions, such as player selection, substitutions, and tactical adjustments.</p>
--	---	--

<p>3.</p> <p>Game Plan: What AI can do for Football, and What Football can do for AI</p>	<p>Karl Tuyls, Shayegan Omidshafiei, Paul Muller, Zhe Wang, Jerome Connor, Daniel Hennes</p> <p>May 2021, Journal of Artificial Intelligence Research</p>	<p>The paper is a theoretical discussion that sets the stage by emphasizing the unique synergy between artificial intelligence (AI) and the world's most popular sport, football.</p> <p>It acknowledges that football analytics provides a rich playground for AI research due to its complexity, real-world dynamics, and massive data availability.</p> <p>Data-Driven Insights: AI techniques enable data-driven insights into player performance, team strategies, and match outcomes.</p> <p>Predictive Models: Algorithms predict player ratings, injury risks, and even transfer market values.</p> <p>Computer Vision: AI-powered systems track player movements, ball trajectories, and tactical patterns during live matches.</p> <p>Fan Engagement: Personalized content, interactive apps, and augmented reality experiences enhance fan engagement.</p> <p>The paper discusses the following Challenges in Football Analytics:</p> <p>High Dimensionality: Football data is multidimensional, including player attributes, match events, and contextual factors.</p> <p>Temporal Dependencies: Understanding how events unfold over time is crucial.</p> <p>Noise and Uncertainty: Real-world data is noisy, incomplete, and subject to various uncertainties.</p> <p>Statistical Learning Techniques:</p> <p>Regression Models: Predicting continuous variables (e.g., player performance metrics).</p>
---	---	---

		<p>Classification Models: Identifying player positions (e.g., forward, midfielder, defender).</p> <p>Clustering: Grouping similar players based on playing style.</p> <p>Time Series Analysis: Capturing temporal patterns in match data.</p> <p>Game Theory Applications:</p> <p>Penalty Kicks: Applying game theory to analyze optimal strategies for penalty takers and goalkeepers.</p> <p>Player Interactions: Modeling interactions between players as strategic games.</p> <p>Computer Vision Challenges:</p> <p>Player Tracking: Extracting player trajectories from video feeds.</p> <p>Pose Estimation: Inferring player body positions and orientations.</p> <p>Event Detection: Recognizing goals, fouls, and other key events.</p>
<p>4.</p> <p>Multi-stream Data Analytics for Enhanced Performance Prediction in Fantasy Football</p>	<p>Nicholas Bonello, Joeran Beel, Seamus Lawless, Jeremy Debattista</p> <p>2019, 27th AIAI Irish Conference on Artificial</p>	<p>Fantasy Premier League (FPL) performance predictors often rely solely on historical statistical data to predict player performances.</p> <p>However, this approach overlooks external factors such as injuries, managerial decisions, and other tournament match statistics.</p> <p>Traditional statistical predictors lack the ability to incorporate real-time information and human feedback.</p>

	Intelligence and Cognitive Science	<p>Predictions based solely on historical data may not account for dynamic changes during a season.</p> <p>The authors introduce a novel method that enhances performance prediction by automatically incorporating human feedback into the model.</p> <p>They consider multiple streams of data beyond historical statistics to improve predictions.</p> <p>Data Streams Used:</p> <p>Previous Performances: Historical player performance data.</p> <p>Fixture Difficulty Ratings: Upcoming fixture challenges.</p> <p>Betting Market Analysis: Insights from betting odds.</p> <p>General Public and Expert Opinions: Social media discussions, web articles, and expert insights.</p> <p>The proposed model was tested on the English Premier League 2018/19 season.</p> <p>It outperformed regular statistical predictors by over 300 points, averaging 11 points per week.</p> <p>The model ranked within the top 0.5% of players, achieving a rank of 30,000 out of over 6.5 million players.</p> <p>By incorporating diverse data streams and human feedback, this approach provides more robust and accurate predictions.</p> <p>It demonstrates the potential of combining statistical analysis with real-world insights for fantasy football enthusiasts</p>
--	---	--

<p>5.</p> <p>Football Player Value Assessment Using Machine Learning Techniques</p>	<p>Ahmet Talha Yiğit, Barış Samak and Tolga Kaya</p> <p>2019, 27th AIAA Irish Conference on Artificial Intelligence and Cognitive Science</p>	<p>Sports analytics is a growing field worldwide, and one of its open problems is the valuation of football players.</p> <p>The aim of this study is to establish a football player value assessment model using machine learning techniques to support transfer decisions by football clubs.</p> <p>The proposed models are mainly based on intrinsic features of individual players, as provided in the Football Manager video game.</p> <p>The individual statistics of 5316 players active in 11 different major leagues from Europe and South America serve as the dataset.</p> <p>The study employs advanced supervised learning techniques:</p> <p>Ridge and Lasso Regressions: Regularized linear regression models.</p> <p>Random Forests: Ensemble models combining decision trees.</p> <p>Extreme Gradient Boosting (XGBoost): Gradient boosting algorithm.</p> <p>All models are built in the R programming language.</p> <p>The models' performances are compared based on their mean squared errors.</p> <p>An ensemble model with inflation is proposed as the output.</p> <p>The goal is to value players based on their normalized abilities, relatively free from environmental variables.</p> <p>The model aims to provide better results than current models relying solely on in-field game statistics.</p>

		<p>Considering the expansion of the football industry, a model using the latest developments in data analytics and machine learning addresses a significant problem for the industry.</p> <p>It can be a valuable financial leverage for clubs seeking to expand their successes and profits.</p> <p>This research contributes to the world of football by providing a data-driven approach to player valuation.</p> <p>By leveraging machine learning techniques, clubs can make more informed transfer decisions.</p>
<p>6.</p> <p>Fantasy Premier League Performance Prediction</p>	<p>Pratik Pokharel, -Arun Timalina, Sanjeeb Panday, Bikram Acharya</p> <p>2022, 12th IOE Graduate Conference</p>	<p>This paper proposes a rational approach to player selection, team drafting, and transfers by predicting ROI using xgboost regression.</p> <p>The study also evaluates the impact of fixture congestion on FPL points using mid-week cup fixture data.</p> <p>Evaluation based on FPL global ranking reveals that initial drafted teams without transfers performed better than those with transfers, which were hindered by dependency on the accuracy of the regression model.</p> <p>The mean RMSE score for all players was 2.048, and the effect of cup fixture congestion on FPL points was found to be insignificant.</p> <p>The uncertainty of team selection makes it challenging for FPL managers, with Game-week 1 being crucial for laying the foundation of the season.</p> <p>The paper highlights the vast number of potential team combinations and the difficulty in making optimal selections.</p> <p>FPL analytics traditionally rely on historical statistical data but may overlook external factors such as mid-week fixture fatigue and squad</p>

		rotation strategies employed by managers due to European and domestic cup competitions.
7. Who Should Be the Captain This Week? Leveraging Inferred Diversity-Enhanced Crowd Wisdom for a Fantasy Premier League Captain Prediction	Shreyansh Bhatt, Keke Chen, Valerie L. Shalin, Amit P. Sheth, Brandon Minnery 2019, 13th International AAAI Conference on Web and Social Media (ICWSM)	Utilizes the Wisdom of Crowd effect to predict productive players in Fantasy Sports. Proposes the SmartCrowd framework to select a small, smart crowd using participants' Twitter posts. Three main steps: characterizing participants using summary word vectors clustering participants based on these vectors sampling participants from clusters to form diverse crowds. Empirical evaluation on the Fantasy Premier League (FPL) captain prediction problem shows SmartCrowd outperforming random crowds and crowds consisting of top experts identified from previous performance data. Social media-based diversity supports the sampling of smarter crowds that collectively predict productive players. Studies the assembly of a small subset of the crowd using semantic diversity inferred from participants' Twitter posts . Explores the weekly captain selection task in Fantasy Premier League (FPL) starting 11 player teams. Utilizes crowd wisdom for determining parameters influencing captain choice. Introduces the SmartCrowd approach, extending to other prediction problems, based on social media posts.

		<p>Mines FPL user tweets to infer crowd diversity based on topic and communication patterns.</p> <p>Represents each Twitter user by a collection of their FPL tweets and summarizes them using Word2vec.</p> <p>Tests multiple clustering strategies and selects optimal representatives from clusters using multi-objective optimization.</p> <p>Evaluates the approach using captain picks, points earned, and participants' previous seasons' performance scores.</p> <p>Investigates questions related to the effectiveness of semantic analysis, diversity-based crowd selection, comparison with expert crowds, and the impact of diversity and crowd size.</p>
<p>8.</p> <p>Time Series Modeling for Dream Team in Fantasy Premier League</p>	<p>Gupta, Akhil</p> <p>2017, International Conference on Sports Engineering (ICSE)</p>	<p>Utilizes data from the official Fantasy Premier League website for the past five to six years.</p> <p>Cross-checks historical data using Kaggle kernels to ensure accuracy.</p> <p>Employs data scraping techniques in Python 3 to collect two types of datasets: Master FPL dataset and Points dataset for each season.</p> <p>Identifies and handles missing values in the datasets.</p> <p>Scales down the cost field in the Master FPL dataset.</p> <p>Creates a new ID field for matching datasets and merges points data into a single dataframe.</p> <p>Removes players with incomplete historical data and those who left the league, ensuring data consistency.</p> <p>Utilizes historical data to predict future player performance.</p>

		<p>Implements two separate models: ARIMA and LSTM-RNN, which are later ensembled for improved accuracy.</p> <p>Treats each player's performance data as a separate time series.</p> <p>Validates models using data from previous seasons.</p> <p>Formulates the problem of selecting the dream team as a linear programming (LP) problem.</p> <p>Seeks to maximize the total points of the selected players within budget constraints.</p> <p>Utilizes the Pulp library in Python 2.7 for solving the LP problem.</p> <p>Automates the prediction process after setting optimal parameters.</p> <p>Validates models using RMSE and compares predicted values with actual values.</p> <p>Determines optimal ensemble proportions for ARIMA and LSTM-RNN models.</p> <p>Identifies the dream team for the upcoming season based on the optimized predictions.</p> <p>Analyzes additional features such as Points per Match (PPM), Cost per Point (CPP), and Cost-point index (CPI) to evaluate player performance.</p> <p>Discusses factors influencing player performance and team selection.</p> <p>Highlights insights gained from the study, including player loyalty, nationality distribution, and team performance analysis.</p> <p>Tracks the performance of the selected dream team for validation and further refinement of the approach.</p>
--	--	---

9.		
<p>Football players performance analysis and formal/informal media: Sentiment analysis and semantic similarity</p>	<p>Gustavo Henrique Sousa Silva, Rui Jorge Henriques Calado Lopes</p> <p>2021, ISCTE</p>	<p>The study utilized data from informal media (e.g., Reddit) and formal media (e.g., Live Match commentary, Player Ratings) to compare semantic similarity with key phrases from Work Domain Analysis (WDA).</p> <p>Semantic Analysis involved comparing WDA key phrases with media entries using BERT to generate vector representations for textual data was used. Cosine similarity was computed between these representations.</p> <p>Comments Processing: Each comment was subdivided into sentences.</p> <p>Similarity Analysis: BERT was used to compare sentences with WDA key phrases, producing similarity scores ranging from 0 to 1.</p> <p>Output Generation: Heatmaps were created to visualize the similarity values obtained.</p> <p>Linguistic Analysis: Named entities and proper nouns were recognized to understand the main subjects of the match.</p> <p>Sentiment Analysis: Polarity and subjectivity of comments were analyzed using TextBlob and Stanza libraries.</p> <p>Output Generation: Polarity and subjectivity values were expressed in the range of -1 to 1 and 0 to 1, respectively.</p> <p>The study employed Semantic Analysis to compare media entries with WDA key phrases and Sentiment Analysis to analyze the polarity and subjectivity of comments. These methodologies provided insights into the perception of football matches across different media sources.</p>
10.		

<p>TF-Pundit: A Real-time Football Pundit based on Twitter</p>	<p>Karthik Anantha Padmanabhan</p>	<p>Actor Model Implementation: The system, named "TF-Pundit," is implemented using the Actor model, which facilitates concurrent computation. Actors receive messages and make decisions based on them, allowing the system components to execute concurrently.</p> <p>Text Processing Components:</p> <p>Performance Analyzer: Analyzes sentiment associated with players based on aggregated tweets. Utilizes a lexicon-based classifier to assign sentiment scores to players, considering words specific to football.</p> <p>Summarizer: Summarizes football events based on tweets, discarding subjective opinions and selecting objective statements.</p> <p>Performance Analyzer Evaluation: Compares ratings assigned by TF-Pundit with those from Goal.com, using a scaling formula. Scores are scaled between 0 and 5.</p> <p>Summarizer Evaluation: Manually assigns scores to summaries on a scale of 1-10 based on how well they summarize one-minute intervals of football events.</p> <p>The current implementation assumes that sentiment expressed in a tweet is solely for the mentioned player, which may not always be accurate. Tweets often express sentiments about multiple players simultaneously, leading to inaccuracies in sentiment analysis.</p> <p>Generally, TF-Pundit's ratings were not significantly different from Goal.com ratings, indicating reasonable accuracy in assessing player performances.</p> <p>Instances where discrepancies occurred were analyzed:</p>
---	---	---

		<p>For example, a significant difference in the rating for "Shaarawy" was attributed to a specific event during the game that led to an inflated score due to numerous positive tweets.</p> <p>This highlights the system's sensitivity to specific events and the need for context-aware analysis.</p> <p>Prominent games, characterized by a higher volume of tweets and more noise, posed greater challenges for accurate player performance assessment.</p>
--	--	---

2.2 GAPS IDENTIFIED

The literature review conducted revealed a gap in existing research pertaining to Fantasy Premier League (FPL) analysis. Specifically, none of the reviewed papers addressed the integration of past season performance analysis coupled with the consideration of team versus team history, which provides insight into opponent strength when assessing player performance. Furthermore, sentiment analysis, a crucial aspect in understanding the contextual nuances surrounding player performance, was not explored in any of the identified literature. These findings underscore the opportunity to contribute novel insights and methodologies to the field of FPL analytics, particularly by incorporating historical performance data and sentiment analysis into predictive models for enhanced decision-making capabilities among FPL managers.

2.3 PROBLEM STATEMENT

Creating a Fantasy Premier League (FPL) Assistant that optimizes team selection for FPL managers, considering insights from player performance in past sessions and already played matches, budget constraints, fixture prioritization, and sentiment analysis of players. This assistant aims to maximize the winning possibility of FPL managers by efficiently utilizing available resources and providing personalized recommendations tailored to individual team compositions and preferences.

3. TECHNICAL SPECIFICATION

3.1 REQUIREMENTS

3.1.1 FUNCTIONAL

1. **Guided Learning Models:**

- The system should provide guided recommendations for team optimization based on machine learning models.
- Machine learning algorithms should analyze historical player performance data and suggest optimal team configurations for upcoming matches.

2. **Social Media Sentiment Analysis:**

- Integration with social media platforms to gather sentiment analysis data on FPL players.
- Sentiment analysis should provide insights into the popularity and sentiment surrounding FPL players within the community.

3. **Fixture Prioritization:**

- The system should prioritize fixtures based on team form, fixture difficulty ratings, and player fatigue.
- Users should be provided with fixture analysis and recommendations to optimize team selection.

4. **Budget Management:**

- Budget constraints should be enforced to ensure users adhere to FPL budget limits.
- The system should suggest player transfers and substitutions to maximize team performance within budget constraints.

5. **Real-time Updates:**

- Real-time updates on player performance, injuries, and other relevant news should be provided to users.
- Users should be alerted to make informed decisions regarding team management.

3.1.2 NON-FUNCTIONAL

1. **Performance:**

- The system should be responsive and provide fast performance, even during peak usage times.
- Response times for recommendations and updates should be minimal to enhance user experience.

2. **Scalability:**

- The system should be able to handle a large number of concurrent users without degradation in performance.
- Scalability measures should be implemented to accommodate growth in user base and data volume.

3. **Security:**

- User data, including personal information and login credentials, should be encrypted and stored securely.
- Authentication mechanisms should be robust to prevent unauthorized access to user accounts.

4. **Reliability:**

- The system should be highly reliable, with minimal downtime and errors.
- Backup and recovery mechanisms should be in place to ensure data integrity and availability.

5. **Usability:**

- The user interface should be intuitive and user-friendly, catering to users with varying levels of technical expertise.
- Help and support features should be available to assist users in navigating the system.

6. **Privacy:**

- User privacy should be protected, and data handling practices should comply with relevant privacy regulations.

- Users should have control over their data and be able to manage privacy settings.

7. **Compatibility:**

- The system should be compatible with a range of devices and web browsers to ensure accessibility for all users.
- Compatibility with mobile devices should be prioritized to support users on-the-go.

3.2 FEASIBILITY STUDY

3.2.1 TECHNICAL FEASIBILITY

1. **Data Availability and Quality:** Assess the availability and quality of data required for implementing guided learning models and social media sentiment analysis. Ensure that sufficient and relevant data sources are accessible to train machine learning models and perform sentiment analysis effectively.
2. **Computational Resources:** Evaluate the computational resources needed to process large datasets and run machine learning algorithms efficiently. Consider factors such as server capacity, processing speed, and memory requirements to ensure the system can handle the computational demands of the project.
3. **Integration Capabilities:** Determine the feasibility of integrating with external APIs or platforms to access social media data and sentiment analysis tools. Ensure that APIs are well-documented, stable, and provide the necessary functionality for data retrieval and analysis.
4. **Scalability:** Assess the scalability of the system architecture to accommodate potential growth in user base and data volume over time. Design the system with scalability in mind, leveraging cloud-based solutions or scalable infrastructure components to handle increased load and demand.

3.2.2 ECONOMIC FEASIBILITY

1. **Cost of Development:** Estimate the development costs associated with building the proposed system, including software development, data acquisition, and integration expenses. Consider the costs of hiring skilled personnel, acquiring

necessary software licenses, and purchasing hardware infrastructure.

2. **Operational Costs:** Evaluate the ongoing operational costs of maintaining and running the system, including hosting fees, maintenance expenses, and any recurring subscription costs for external services or APIs. Ensure that the project remains financially viable and sustainable in the long term.
3. **Return on Investment (ROI):** Assess the potential return on investment from the project, considering factors such as increased user engagement, subscription revenue, or advertising opportunities resulting from a successful implementation. Conduct a cost-benefit analysis to determine whether the expected benefits justify the investment.

3.2.3 SOCIAL FEASIBILITY

1. **User Acceptance:** Evaluate the potential acceptance and adoption of the system by users, including FPL enthusiasts and fantasy sports fans. Conduct user surveys or focus groups to gather feedback on the proposed features and functionalities, ensuring that the system meets user expectations and preferences.
2. **Community Engagement:** Assess the potential impact of social media sentiment analysis on fostering community engagement and interaction among FPL managers. Explore how sentiment analysis insights can enhance the user experience, facilitate discussions, and build a sense of community within the FPL ecosystem.
3. **Ethical Considerations:** Consider ethical implications related to data privacy, user consent, and responsible use of social media data in sentiment analysis. Ensure that the project adheres to ethical guidelines and respects user privacy rights, building trust and credibility among users and stakeholders.

3.3 SYSTEM SPECIFICATION

3.3.1 HARDWARE SPECIFICATION

1. **Memory (RAM):** Sufficient RAM to accommodate data processing and machine learning model training tasks, depending on the size of the dataset and complexity of the models.

2. Storage: Adequate storage space to store datasets, trained models, and other project-related files. Consider using scalable storage solutions to accommodate potential growth in data volume.
3. Processing Power: Multi-core processors or high-performance CPUs to support parallelized data processing and model training tasks, reducing processing time and improving efficiency.

3.3.2 SOFTWARE SPECIFICATION

1. Programming Languages:

- Python: For data analysis, machine learning model development, and web application development using frameworks like Flask or Django.

2. Data Analysis and Machine Learning Libraries:

- Pandas: For data manipulation and analysis.
- NumPy: For numerical computing and array operations.
- Scikit-learn: For machine learning model development and evaluation.
- Statsmodels: For time series analysis and statistical modelling.

3. Development Tools:

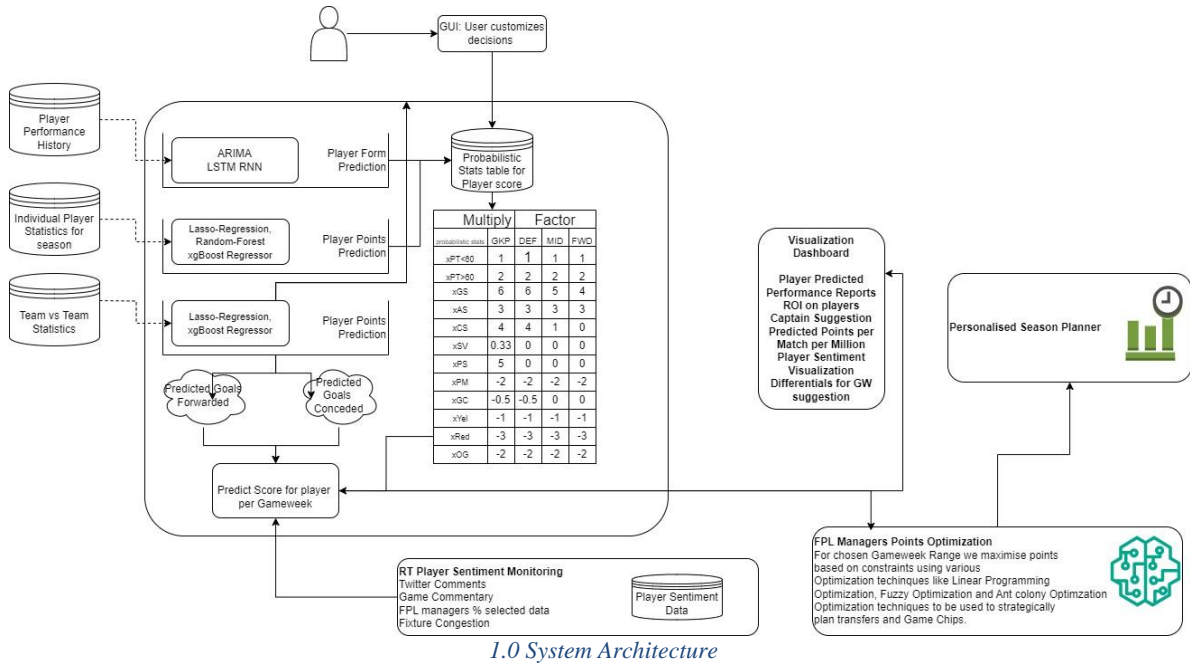
- Jupyter Notebook or IDEs like PyCharm, VS Code, or Atom: For writing and executing Python code, data exploration, and model development.
- Git: For version control and collaboration among team members.

4. External APIs and Libraries:

- Twitter API or sentiment analysis libraries (e.g., NLTK, TextBlob, VADER): For accessing social media data and performing sentiment analysis, as mentioned in the project scope.

4. DESIGN APPROACH AND DETAILS

4.1 SYSTEM ARCHITECTURE



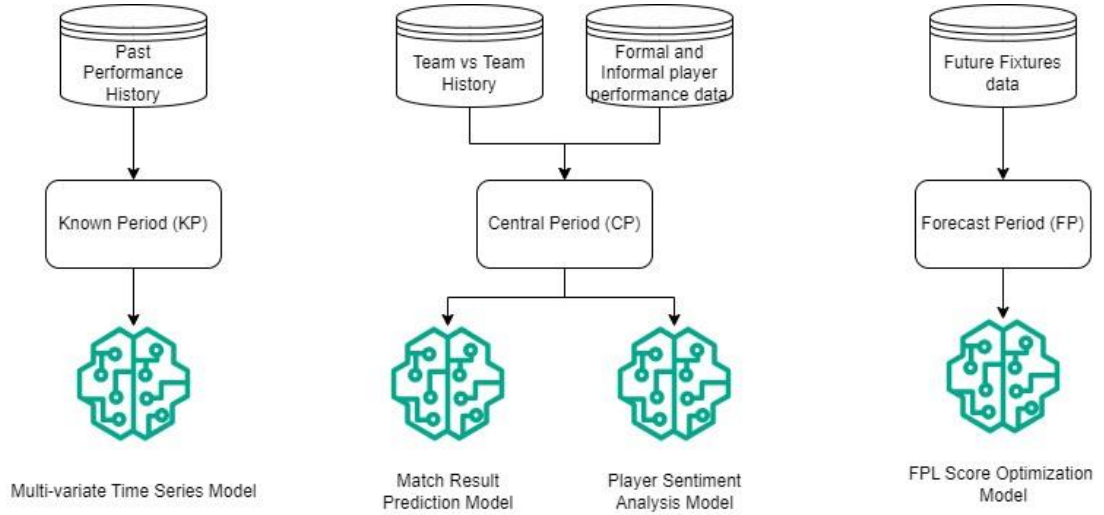
Before each Game week, the Model plans for future Game week fixtures with decreasing order of priority to each successive fixture, and for each player calculates Team Form and Predicted Team Scoring Index (xGfor) and Predicted Team Conceding Index (xGagnst) by using Form, Attack Index, Defence Index, Fixture Difficulty Rating FPL API data, previous history from FPLAnalytics.com and whoscored.com, time-series forecasting from previous season data and team fatigue Index from other Fixtures from whoscored.com.

Now, if the player is an Attacker/ Midfielder, we take the Predicted Team Score Index (xGfor) and the probable minutes per 90 index (xMP) and the match fitness (fatigue/ rest probability) and the Form of the player and the expected Scoring Index (xSc) and the expected Assisting Index (xAst) to calculate the probable points for the game (xPts).

If the player is Defender/ Goalkeeper, we take the inverse of Predicted Team Conceding Index (xGagnst) and the probable minutes per 90 index (xMP) and the match fitness (fatigue/ rest probability) and the Form of the player and the expected save Index (xSv) and the discipline Index to calculate the probable points for the game.

For each player, we also consider a player Selection popularity Model where we using

% selected, % captained data from FPLAnalytics and FPL API and using Twitter Sentiment Analysis on tweets on threads related to @FPLtweets for Sentiment analysis for players and formulate Popularity Metric using BERT (Bidirectional Encoder Representations from Transformers)



1.1 System Overview

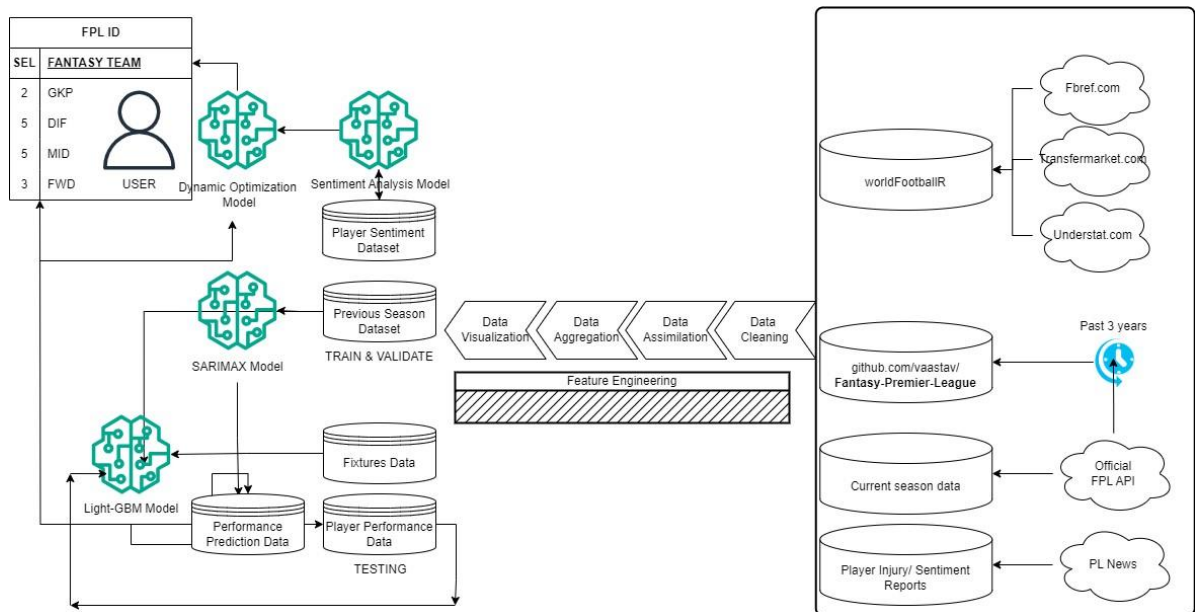
We then design a statistical model that predicts the best permutations of team (playing XI + 4 bench players) with formation using Predicted points per Cost Unit (Million) and create best possible team that maximizes the score for the upcoming fixtures.

For Time-Series Modelling we will be using a hybrid of Autoregressive Integrated Moving Average (ARIMA) known as SARIMAX to account for the seasonality for time series prediction and effect of exogenous regressors of player stats and subsequent maximization of total points using Linear Programming (LPP).

For Prediction of Fixture results and points for same season we will be using Random Forest Regressors with extreme Gradient Boosting (xgBoost). We have chosen LightGBM as an improvement over xgboost as LightGBM uses a novel technique of Gradient-based One-Side Sampling (GOSS) to filter out the data instances for finding a split value while XGBoost uses pre-sorted algorithm & Histogram-based algorithm for computing the best split. We aim to feed trained data to an LLM model to create a chat interface for user with the data.

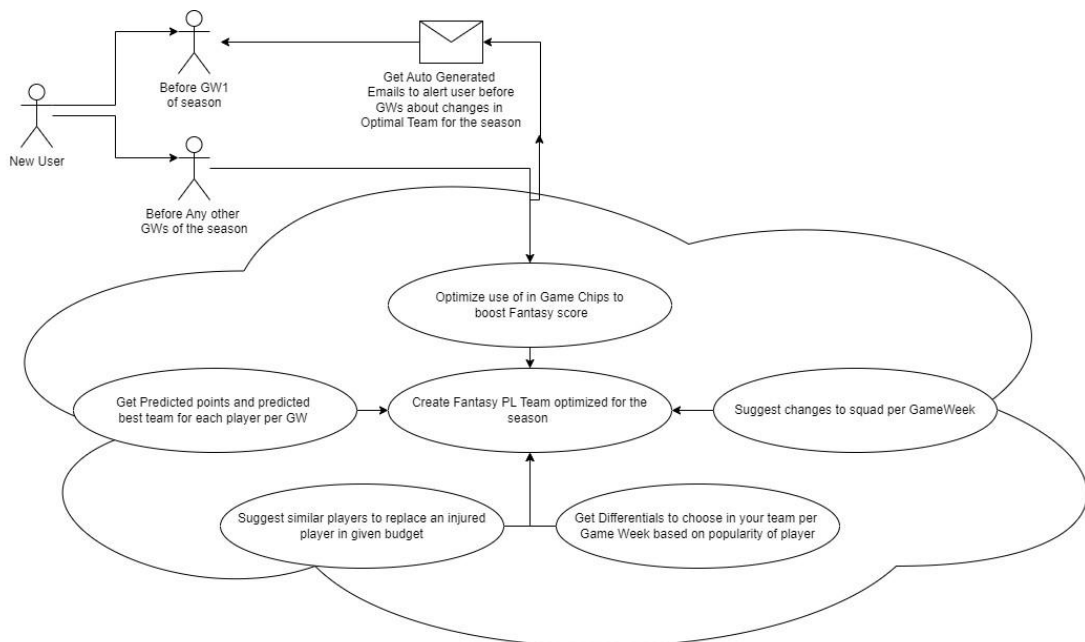
4.2 DESIGN

4.2.1 Data Flow Diagram



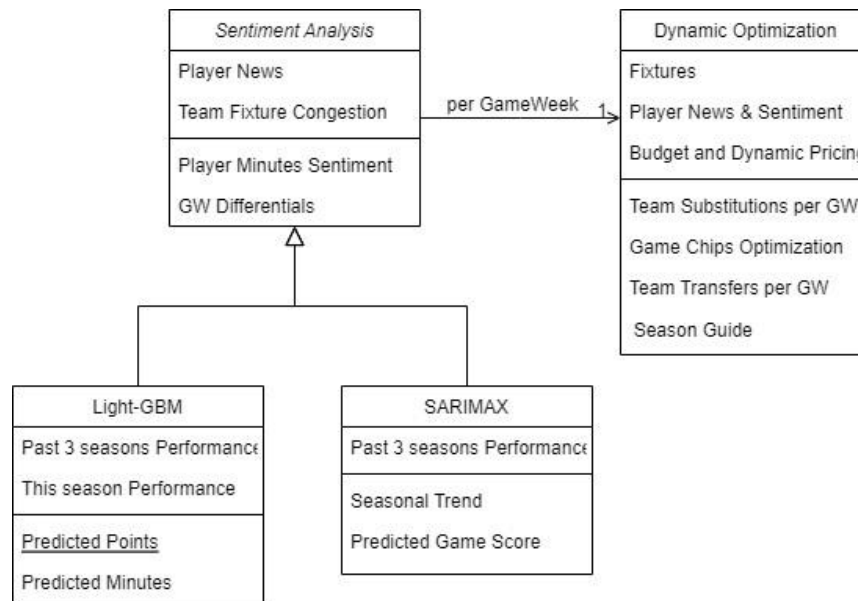
1.2 Dataflow

4.2.2 Use Case Diagram



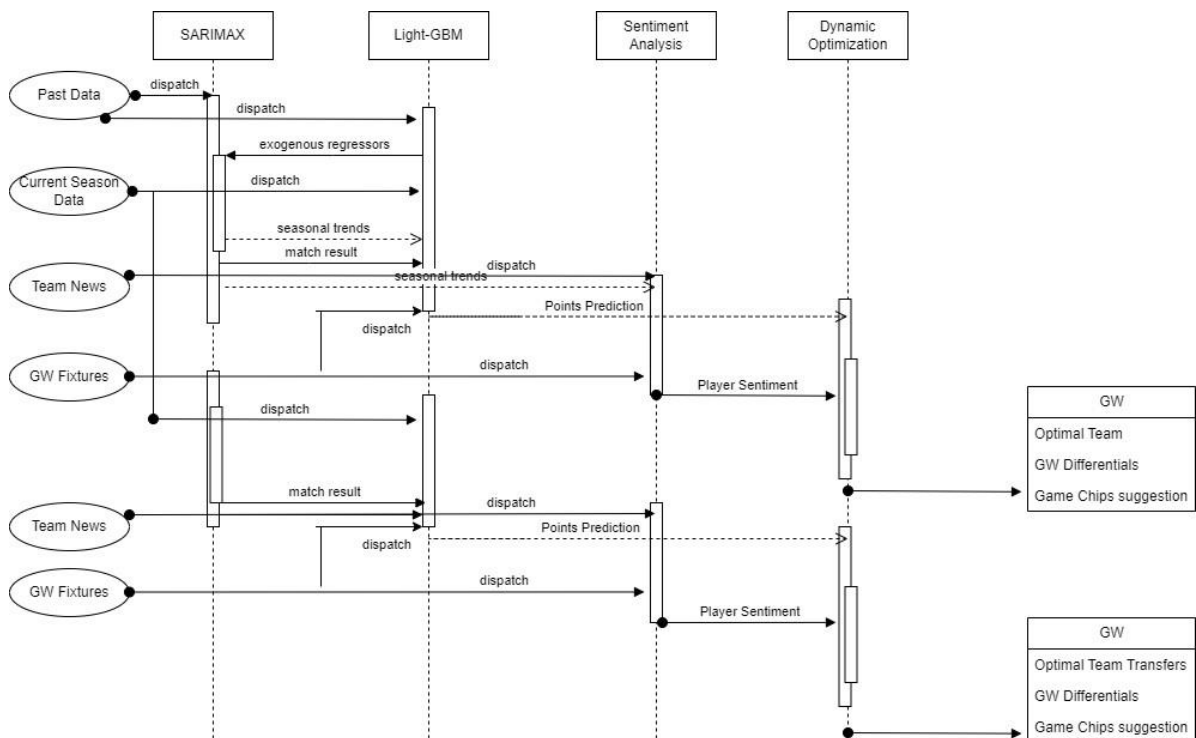
1.3 Usecase

4.2.3 Class Diagram



1.4 Class

4.2.4 Sequence Diagram



1.5 Sequence

4.3 CONSTRAINTS, ALTERNATIVES AND TRADEOFFS

1. **Data Quality and Availability:** We are aware of the difficulties in obtaining the high-quality and readily available data sources needed to produce precise forecasts and analyses. We seek to overcome these limitations by applying strict data pretreatment and validation procedures. The limits include incomplete or unreliable data, limited history records for specific players or teams, and potential biases in the data.

2. **Computing Resources:** We draw attention to the computing resources needed to handle massive data sets, develop real-time machine learning models, and produce predictions. It's crucial to strike a balance between computing efficiency and model complexity, particularly when scaling up platform deployment to support an expanding user base.

3. **Model Accuracy vs. Interpretability:** We take into consideration the trade-off between interpretability and model correctness. Even though more intricate machine learning models might have better prediction accuracy, users might find it more difficult to understand and comprehend them. To guarantee that customers can trust and comprehend the recommendations made, we strive to strike a balance between transparency and accuracy in the model's architecture.

4. **Budget Constraints:** Our technology takes into account the financial limitations that FPL managers have and provides suggestions for the best squad selection while staying within these constraints. Nonetheless, we are aware of the trade-offs of choosing expensive, high-performing players over less expensive options, and we work hard to offer user preferences-based suggestions that are well-balanced.

5. **User Interface Design:** We address the trade-offs in user interface design between providing comprehensive data and insights versus maintaining simplicity and ease of use. Prioritizing information and features based on user preferences and feedback is crucial to ensuring an intuitive and engaging user experience.

6. **Real-time Data Updates:** We talk about the difficulties and trade-offs involved in incorporating real-time data streams into the platform, including player injuries or lineup modifications. A crucial factor in platform design is striking a balance between the requirement for quick updates, processing complexity, and potential delays in data availability.

7. **Ethical and Legal Considerations:** We draw attention to the moral and legal limitations concerning user consent, data privacy, and fair use, particularly when utilizing third-party data sources or sentiment analysis from social media. In designing and implementing our platform, user trust and compliance are of utmost importance.

5. SCHEDULE, TASKS AND MILESTONES

5.1 GANTT CHART

GANTT CHART



1.6 Gantt Chart

5.2 MODULE DESCRIPTION

5.2.1 Data Scrapping:

- worldfootballR :

FBref.com - a whole host of data to analyse, including results, match stats, season long stats, player and team stats, etc.

Transfermarkt.com - player market values, team transfer history, player transfer history.

Understat.com - shot locations data for matches played in the major leagues.

- Fantasy Premier League API :

<https://fantasy.premierleague.com/api>

- Github Repository fir Premier League previous year statistics:
<https://github.com/vaastav/Fantasy-Premier-League>

5.2.2 Data Cleaning

- Removing Duplicates:

Involves identifying and eliminating duplicate records within the dataset to ensure data integrity and accuracy.

For our project, removing duplicates ensures that each player's performance statistics are uniquely represented, preventing any bias or distortion in the analysis and modeling process.

- Interpolating to Fill Missing Data:

Imputes missing values in the dataset by estimating them based on the surrounding data points.

For our project, interpolating missing data helps ensure completeness and consistency in player performance records, allowing for more comprehensive analysis and modeling.

- Mapping Teams to Team ID:

Establishes a relationship between team names and unique team identifiers (IDs) for easy referencing and data consistency.

In our project, mapping teams to team IDs facilitates uniformity in representing team information across different datasets and sources, enhancing data integration and analysis capabilities.

- Mapping Players to Player ID:

Associates player names with unique player identifiers (IDs) to standardize player references and enable seamless data linking and analysis.

This mapping ensures that player performance data from various sources can be effectively integrated and analyzed within the project framework.

- Merging Season Stats:

Combines player performance statistics from different seasons into a unified dataset, allowing for comprehensive analysis and trend identification across multiple seasons.

Merging season stats involves aggregating player performance metrics such as goals scored, assists, clean sheets, etc., from individual seasons into a single dataset. This aggregated dataset provides a holistic view of player performance over time, enabling the development of more robust models and strategies for FPL team optimization. Additionally, by merging season stats, we can identify patterns, trends, and player consistency across multiple seasons, offering valuable insights for FPL managers in team selection and strategy planning

5.2.3 Data Assimilation

1. Merging Opponent Data:

- Involves combining information about opponent teams with player performance data to enrich the dataset.
- This merging enables the analysis of player performance in the context of the opponents they faced, providing insights into player performance variations based on the strength of the opposition.

2. Creating Mapping Between Tables:

- Establishes relationships between different tables or datasets by defining key identifiers that link corresponding records.

- This mapping facilitates data integration and retrieval, allowing for seamless analysis and querying across multiple tables within the database.
3. Converting from Raw API Data to Structured Data:
- Refers to the process of transforming raw data obtained from APIs into a structured format suitable for analysis and storage.
 - This conversion involves parsing, cleaning, and organizing the raw data into tables or datasets with well-defined schemas, making it easier to analyze and extract insights.
4. Combining Data Collected from Various Sources:
- Integrates data obtained from diverse sources, such as APIs, databases, files, or web scraping, into a unified dataset.
 - By combining data from multiple sources, we can leverage complementary information to enrich analysis, gain a comprehensive understanding of the subject matter, and uncover hidden patterns or correlations.

5.2.4 Feature Engineering

- Last N Week Stats Calculation:
Implemented in functions `get_last_stats`, `get_players_last_stats_test`, and `get_all_players_last_stats`. Calculates statistics for each player from the last N gameweeks. The statistics considered are specified in the `history_stats` list.
- Mean and Standard Deviation Calculation:
 - Implemented in function `create_features`.
 - Calculates the mean and standard deviation of selected statistics (`mean_features` and `std_features`) from the last N gameweeks.
 - Appends these mean and standard deviation features to the

dataframe.

- Percentage Value to Team and Position Rank Calculation:
 - Implemented within the loop iterating over each gameweek.
 - Calculates the percentage value of each player relative to the total value of their team.
 - Calculates the position rank of each player within their position and team based on value.
- Opponent Team and Last Season Position Extraction:
 - Implemented in the loop iterating over each year and gameweek.
 - Extracts the opponent team for each match and their position in the previous season.
 - Merges this information with the main dataframe.

5.2.5 Model Training

1. SARIMAX - Seasonal AutoRegressive Integrated Moving Average with eXogenous factors is a time series forecasting model that extends the traditional ARIMA model to handle seasonality and exogenous variables.

$$d_t = c + \sum_{n=1}^p \alpha_n d_{t-n} + \sum_{n=1}^q \theta_n \epsilon_{t-n} + \sum_{n=1}^r \beta_n x_{n_t} + \sum_{n=1}^P \phi_n d_{t-sn} + \sum_{n=1}^Q \eta_n \epsilon_{t-sn} + \epsilon_t$$

2.1 Sarimax

- Discrete Fourier Transform testing has been conducted. DFT testing is a method used to analyse periodic patterns or seasonality in time series data. By completing DFT testing, the model has assessed the presence and characteristics of seasonality in the data.
- Model data assimilated implying that the model has incorporated and integrated relevant data. This likely includes historical time series data, exogenous variables (if applicable), and any other relevant

information used in model training and analysis.

2. Light GBM - Light Gradient Boosting Machine is a powerful and efficient gradient boosting framework used for supervised learning tasks such as classification, regression, and ranking. It is designed to be highly scalable and can handle large datasets with millions of instances and features. Gradient boosting framework that uses tree-based learning algorithms. It is designed to be distributed and efficient with the following advantages: Faster training speed and higher efficiency, Lower memory usage, better accuracy, Support of parallel, distributed, and GPU learning and Capable of handling large-scale data.
 - Trained on Last 3 seasons
 - Trained on First 10 Game weeks for the new season

Data Preprocessing:

- combines the "name" and "kickoff_time" columns to create a unique index for each data point in both the training and test datasets.
- drops duplicate rows based on this index, keeping only the last occurrence.
- converts the "kickoff_time" column to datetime format and extracts additional features such as day of the week, month, hour, and week from the datetime data.
- drops irrelevant columns such as "kickoff_time" and "minutes" from both datasets.

Data Encoding:

- encodes categorical variables in the training and test datasets using factorization (converting categorical variables into numerical ones).
- The encoding is applied to all object-type columns except for "team", "name", and "position".

Target and Feature Selection:

- separates the target variable "total_points" along with "GW" and "position" from the training dataset into a separate DataFrame called "target".

- removes the target variable "total_points" and "minutes" from the training and test datasets, as well as other columns specified in "dropped_columns".

Data Splitting:

- splits the training data into training and validation sets (x, val) and corresponding target sets (y, y_val) using train_test_split with a test size of 0.1.

Model Training and Validation:

- initializes a LightGBMRegressor model with hyperparameters specified in the code.
- sets up KFold cross-validation with 8 folds for training and validation.
- trains the LightGBM model on each fold of the training data and evaluates it on the validation set.
- prints the fold number and calculates the root mean squared error (RMSE) for both the training and validation sets on each fold.

Prediction Generation:

- After training the model on all folds, generates predictions for the test set using the trained model.
- The predictions are stored in a DataFrame called "predictions_df" with each column representing predictions from one fold.

Reporting:

- prints the RMSE values for both the validation and training sets across all folds.

Final Output:

It selects and sorts specific rows from the test set by "points" in descending order, and stores the players' names, opponent's last season position, home/away status, points, team, and value to use in the Points optimization.

5.2.6 Squad Optimization for Maximum Points

1. Dynamic Programming

We have defined a function `giveSquad(data, budget)` that takes in two parameters: `data`, which represents the player data including their name, position, team, id, value, and weight (`wt`), and `budget`, which is the budget constraint for selecting the squad.

Inside the function, we initialize variables such as `n` for the length of the data, a dynamic programming dictionary `dp` to store computed results, an empty list `teamSquad` to store the final squad, and another empty list `team` to track the players selected for the squad.

Objective Function:

$$\begin{aligned} \max z = \mathbb{E} \left[\sum_{p \in \mathcal{P}} \sum_{t \in \mathcal{T}} \left(\rho_{pt}(\omega_t) y_{pt} + \rho_{pt}(\omega_t) f_{pt} + \epsilon \rho_{pt}(\omega_t) h_{pt} + \sum_{l \in \mathcal{L}} \kappa_l \rho_{pt}(\omega_t) g_{ptl} \right. \right. \\ \left. \left. + 2 \rho_{pt}(\omega_t) c_{pt} \right) \right] \\ - R \sum_{t \in \mathcal{T}} \alpha_t \end{aligned}$$

2.2 Objective Function

We define a nested recursive function `f(idx, budget, maxGoalKeeper, maxDefender, maxMidFielder, maxForward)` to explore all possible combinations of players within the budget and positional constraints. The function takes parameters such as the current index in the data (`idx`), remaining budget, and maximum allowable players in each position (goalkeeper, defender, midfielder, forward).

The recursive function `f` checks several base cases:

If all position slots are filled (`maxGoalKeeper == 0` and `maxDefender == 0` and `maxMidFielder == 0` and `maxForward == 0`), it returns a score of 0 and the current squad.

If we reach the end of the data (`idx == n`), it checks if any position slots are unfilled; if so, it returns negative infinity as the score and an empty squad. Otherwise, it returns a score of 0 and the current squad.

It checks if the current state (index, budget, positional limits, and selected team) has been computed before and returns the result from the dp dictionary if available.

The core of the function f focuses on selecting players based on their positions and updating the squad accordingly. We consider players one by one and recursively explore two possibilities:

- Pick the current player if it fits within the budget and positional constraints, update the squad and call the function recursively with reduced budget and positional limits.
- Skip the current player and call the function recursively with the same budget and positional limits.

The function then compares the scores obtained from picking and not picking the current player and stores the better result in the dp dictionary.

Finally, the function returns the best score and the corresponding squad based on the computed dynamic programming results.

Overall, this approach efficiently explores all possible combinations of players to determine the best 11 in a 3-5-2 formation within the given budget and positional constraints, optimizing for predicted points.

Constraints:

- Squad Size

To join the game select a fantasy football squad of 15 players, consisting of: 2 Goalkeepers, 5 Defender, 5 Midfielder and 3 Forwards

- Players Per Team: You can select up to 3 players from a single Premier League team.

$$\begin{aligned}
 \sum_{p \in \mathcal{P}^K} x_{pt} &= M^K & t \in \mathcal{T} \\
 \sum_{p \in \mathcal{P}^D} x_{pt} &= M^D & t \in \mathcal{T} \\
 \sum_{p \in \mathcal{P}^M} x_{pt} &= M^M & t \in \mathcal{T} \\
 \sum_{p \in \mathcal{P}^F} x_{pt} &= M^F & t \in \mathcal{T} \\
 \sum_{p \in \mathcal{P}_c} x_{pt} &\leq M^C & t \in \mathcal{T}, c \in \mathcal{C}
 \end{aligned}$$

2.3 Constraints 1.0

- Budget: The total value of your initial squad must not exceed £100 million.

2. Managing Trade-offs to get best Squad for any Gameweek

The `giveOptimized(gw)` function is designed to optimize the selection of a squad for a specific gameweek (gw). It performs the following steps:

- Data Preparation:

It retrieves player data for the specified gameweek using the `giveData(gw)` function and prunes the data to include only relevant columns such as name, position, team, id, points, value, and efficiency.

The pruned data is sorted based on efficiency and points to create two datasets: `effData` (sorted by efficiency) and `optData` (sorted by points).

- Initialization:

It sets up variables such as `teamLimit` (maximum players from a single team allowed in the squad), `budget` (initial budget for player transfers), and a dictionary `teamsDic` to map team names to indices.

- Squad Selection:

It calculates the budget available after including bench players using the `giveBench` function and selects the starting XI using the `giveSquad` function.

If the selected squad has fewer than 11 players, it raises a warning.

- Captain and Vice-Captain Selection:

It identifies the captain and vice-captain based on predicted points and sorts the starting XI accordingly.

It calculates the predicted points for the squad, including the captain's points.

- Actual Points Calculation:

It retrieves the actual points scored by each player in the starting XI and the captain for the specified gameweek (gw) from the `finalDf` dataframe.

It calculates the total actual points and the value of the starting XI based on player values.

- Bench Information:

It includes information about the bench players such as their predicted points, actual points, and total value.

It calculates the remaining budget after deducting the value of the starting XI.

- Return Value:

It returns a list `res` containing information about the selected squad, captain, vice-captain, predicted points, actual points, value of the starting XI, bench information, remaining budget, and the specified gameweek.

3. Optimizing Transfers based on heuristics:

The function `makeChanges(gw, initDf)` is defined to handle transfers for a specific gameweek (`gw`) based on the initial squad data (`initDf`).

It initializes variables and prepares the data for processing, including creating a dataframe `prunedDf` with relevant player information such as points, value, position, etc.

- Player and Squad Details:

It extracts details about the current squad including the starting XI (`curr_XI`), bench players (`curr_bench`), current budget (`curr_budget`), and transfers allowed (`transfer_limit`).

It calculates various metrics such as points, cost, position, and rank for each player in the starting XI and bench.

- Transfer Optimization:

The code optimizes transfers by evaluating potential replacements from the available pool of players (`optDf`), considering factors such as position, points, cost, and rank.

It identifies feasible transfers that comply with squad constraints (e.g., budget, positional limits, team limits) and maximizes the potential points gain.

Transfers are evaluated based on their impact on the squad's predicted points (`points_profit`), new budget (`new_budget`), and overall squad optimization.

- Transfer Execution:

It executes the best transfer based on the calculated optimal transfer (`transfer_out` and `transfer_in`) that maximizes points gain within the constraints.

Updates the squad's starting XI and bench with the new transfers (new_XI and new_bench).

- Performance Evaluation:

Calculates predicted points for the updated starting XI (predPoints), including the captain's points (captain[4]).

Calculates real points based on actual performance data for the gameweek (realPoints).

Computes predicted points for the bench players (predbenchpoints).

- Output and Result Reporting:

Returns a result list (res) containing gameweek, updated starting XI, captain, vice-captain, predicted points, real points, squad value, bench details, predicted bench points, bench value, and new budget.

Returns transfer information (transfer_info) including the transfer out and transfer in details for reporting and analysis.

Overall, the code efficiently manages transfers within the fantasy football game's constraints, optimizing the squad for maximum points gain while considering budget limitations and positional requirements.

Constraints:

- After selecting your squad you can buy and sell players in the transfer market. Unlimited transfers can be made at no cost until your first deadline.
- After your first deadline you will receive 1 free transfer each Gameweek. Each additional transfer you make in the same Gameweek will deduct 4 points from your total score (Classic scoring) and match score (Head-to-Head scoring) at the start of the next Gameweek.

$$q_2 = \underline{Q}$$

$$Ew_t + q_t - \sum_{p \in \mathcal{P}} e_{pt} + \underline{Q} + \alpha_t \geq q_{t+1} \quad t \in T \setminus \{1\}$$

$$\bar{\alpha}(\bar{Q} - q_{t+1}) \geq \alpha_t \quad t \in \mathcal{T}$$

$$q_{t+1} \geq \underline{Q} \quad t \in \mathcal{T}$$

$$q_{t+1} \leq \bar{Q} + (\underline{Q} - \bar{Q})w_t + (\underline{Q} - \bar{Q})r_t \quad t \in \mathcal{T}$$

2.4 Constraints 2.0

- If you do not use your free transfer, you are able to make an additional free transfer the following Gameweek. If you do not use this saved free transfer in the following Gameweek, it will be carried over until you do. You can never have more than 1 saved transfer.

5.2.7 Sentiment Analysis

The Sentiment Analysis module plays a crucial role in the Fantasy Premier League (FPL) Assistant platform by extracting sentiment from news headlines related to players. Twitter Sentiment analysis being out of question due to twitter API banning the access of tweets anymore.

Leveraging Natural Language Processing (NLP) techniques, this module analyzes textual data scraped from BBC News articles to assess the sentiment surrounding specific players. Here's how the module operates:

Data Acquisition: Utilizing the requests library, the module fetches relevant news articles from the BBC News website based on user-defined criteria, such as player names or keywords related to the Premier League.

Text Processing: The retrieved articles are parsed using the BeautifulSoup library to extract headlines and descriptions. These textual data points serve as input for sentiment analysis.

Sentiment Analysis: Initially, the module employs a simplistic approach utilizing the TextBlob library to assess sentiment polarity and subjectivity. It computes the sentiment polarity of each article, categorizing it as positive, negative, or neutral based on the sentiment score.

Deep Learning Sentiment Classification: Subsequently, the module applies a more advanced sentiment analysis technique using pre-trained BERT (Bidirectional Encoder Representations from Transformers) model. This model is fine-tuned to classify text into sentiment categories - positive, negative, or neutral. It tokenizes the combined headline and description texts, feeds them into the BERT model, and extracts predicted sentiment labels.

Aggregated Sentiment Calculation: The module computes the aggregated sentiment polarity by calculating the average sentiment polarity across all analyzed articles. This aggregated sentiment provides a consolidated understanding of the overall sentiment surrounding specific players within the FPL community.

Data Storage: Finally, the module stores the sentiment analysis results, including player names and their corresponding aggregated sentiment polarities, in a CSV file named `sentiment_data.csv`. This allows for further analysis and visualization of sentiment trends over time.

5.2.8 Frontend Development

The frontend module serves as the visual interface for the Fantasy Premier League (FPL) Assistant platform, facilitating user interaction and data presentation. Leveraging React.js and the React Alice Carousel component, the frontend dynamically displays FPL team data fetched in JSON format. Key features include:

Dynamic Data Presentation: Utilizing React.js, the frontend dynamically renders FPL team data in a carousel-like format, providing a visually engaging user experience.

Player Information Display: The frontend presents detailed player information for each game week, including captaincy status, predicted and actual points, total team cost, and bench details.

Player Visualization: Visual representations of players on a football field image provide a clear overview of team composition, aiding managers in strategic decision-making.

Interactive Features: Interactive tooltips enable users to view additional player details such as team affiliation, predicted points, and cost, enhancing user engagement and decision-making capabilities.

Transfer Visualization: Transferred players for each game week are visually represented with an intuitive swap icon, allowing users to track transfer activity efficiently.

5.3 TESTING

5.3.1 Unit Testing

Module: Point Prediction Module

Overview:

This report presents the results of unit testing conducted on the Point Prediction Module. The purpose of these tests is to validate the functionality and correctness of individual components within the application.

Test Environment:

Programming Language: Python 3.9

Testing Framework: pytest 6.2.4

Browser: Google Chrome

IDE: Kaggle Jupyter Notebook Environment

Test Cases and Coverage:

Total Test Cases: 5

Test Coverage: 5/38

Test Results:

Passed: 5 with

Mean RMSE on test set for each Test Case as [1.7339600968841615, 1.668365224381373, 1.721015739246082, 1.6881309613008646, 1.740048919002671]

Skipped: 0

Code Quality Metrics:

Cyclomatic Complexity: Average complexity per module is High. However it is within acceptable limits.

Bug Tracking:

Issue 1: Automated Process reaches timeout. Fix: Store Data for GameWeek predictions locally.

Recommendations and Conclusions:

Separate storage for Predictions locally.

Overview:

This report presents the results of unit testing conducted on the Squad Optimization Module. The purpose of these tests is to validate the functionality and correctness of individual components within the application.

Test Environment:

Programming Language: Python 3.9

Testing Framework: pytest 6.2.4

Operating System: Windows 10

IDE: VSCode Jupyter Notebook Environment

Test Cases and Coverage:

Total Test Cases: 4

Test Coverage: 4/38

Test Results:

Passed: 4 with

Predicted Points for GWs as [73.39600968841615, 113.8365224381373, 72.1015739246082, 68.81309613008646]

Actual Points for GWs as [91, 104, 67, 81]

4/4 of which were above 90 percentile scores for FPL Managers.

Skipped: 0

Code Quality Metrics:

Cyclomatic Complexity: Average complexity per module is Very High. However, without WildCard Optimization, it is within acceptable limits.

Bug Tracking:

Issue 1: Automated Process reaches timeout for GW 7. Fix: Problem in Data as no matches in GW7.

Recommendations and Conclusions:

Run Time within acceptable standards, when Wildcard Optimization is ignored.

Module: Frontend

Overview:

This report presents the results of unit testing conducted on the Frontend. The purpose of these tests is to validate the functionality and correctness of individual components within the application.

Test Environment:

Programming Language: Python 3.9

Testing Framework: pytest 6.2.4

Operating System: Windows 10

IDE: VSCode Jupyter Notebook Environment

Test Cases and Coverage:

Total Test Cases: 1

Test Coverage: NA

Test Results:

Passed: 1

Skipped: 0

Code Quality Metrics:

Cyclomatic Complexity: Average complexity per module is low and within limits.

Bug Tracking:

None

Recommendations and Conclusions:

Run Time within acceptable standards, missing data handled well.

More interactive and Dynamic possible.

5.3.2 Integration Testing

This report presents the results of integrated testing conducted on the FANTASY

PREMIER LEAGUE TEAM OPTIMAZATION USING GUIDED LEARNING MODELS. The purpose of these tests is to validate the interactions and integration between multiple components within the application.

Test Environment:

Programming Language: Python 3.9

Operating System: Windows 10

IDE: VSCode Jupyter Notebook Environment

Test Cases and Coverage:

Total Test Cases: 38

Test Coverage: 100% for Season 22-23

Test Results:

Passed: Given Team performs good enough to beat 90 Percentile Managers Team in FPL.

Gave Best Teams: 38/38

Transferred Player: 36/38

Mean Difference between Actual and Predicted Points for each GW:
4.54448133997855

Season Total Points Accumulated: 3170

GameChips used: 3 out of 4

Skipped: 0

Code Quality Metrics:

Cyclomatic Complexity: Average complexity per module is very High.

Bug Tracking:

NA

Recommendations and Conclusions:

Run Time within acceptable standards, missing data handled well.

6. PROJECT DEMONSTRATION

	name	opponent	last_season_position	was_home	points	team	value
index							
Mohamed Salah	2023-08-13T15:30:00Z	Mohamed Salah	12	1	10.182815	Liverpool	125
Martin Ødegaard	2023-08-12T12:00:00Z	Martin Ødegaard	16	0	6.500668	Arsenal	85
Bukayo Saka	2023-08-12T12:00:00Z	Bukayo Saka	16	0	6.240895	Arsenal	85
Bruno Borges Fernandes	2023-08-14T19:00:00Z	Bruno Borges Fernandes	13	0	5.791867	Man Utd	85
Kevin De Bruyne	2023-08-11T19:00:00Z	Kevin De Bruyne	20	1	5.638430	Man City	105
Gabriel Martinelli Silva	2023-08-12T12:00:00Z	Gabriel Martinelli Silva	16	0	5.130547	Arsenal	80
Marcus Rashford	2023-08-14T19:00:00Z	Marcus Rashford	13	0	4.194368	Man Utd	90
Pascal Groß	2023-08-12T14:00:00Z	Pascal Groß	20	0	4.002104	Brighton	65
Solly March	2023-08-12T14:00:00Z	Solly March	20	0	3.555792	Brighton	65
Eberechi Eze	2023-08-12T14:00:00Z	Eberechi Eze	20	1	3.088195	Crystal Palace	65
Son Heung-min	2023-08-13T13:00:00Z	Son Heung-min	9	1	3.034857	Spurs	90

	name	opponent	last_season_position	was_home	points	team	value
index							
Kieran Trippier	2023-08-12T16:30:00Z	Kieran Trippier	7	0	4.389531	Newcastle	65
Trent Alexander-Arnold	2023-08-13T15:30:00Z	Trent Alexander-Arnold	12	1	4.257807	Liverpool	80
Andrew Robertson	2023-08-13T15:30:00Z	Andrew Robertson	12	1	2.742380	Liverpool	65
Ben Mee	2023-08-13T13:00:00Z	Ben Mee	8	0	2.606454	Brentford	50
Pervis Estupiñán	2023-08-12T14:00:00Z	Pervis Estupiñán	20	0	2.451813	Brighton	50
Virgil van Dijk	2023-08-13T15:30:00Z	Virgil van Dijk	12	1	2.421160	Liverpool	60
Lewis Dunk	2023-08-12T14:00:00Z	Lewis Dunk	20	0	2.408961	Brighton	50
Lisandro Martínez	2023-08-14T19:00:00Z	Lisandro Martínez	13	0	2.341776	Man Utd	50
William Saliba	2023-08-12T12:00:00Z	William Saliba	16	0	2.321176	Arsenal	50
Gabriel dos Santos Magalhães	2023-08-12T12:00:00Z	Gabriel dos Santos Magalhães	16	0	2.318982	Arsenal	50

	name	opponent	last_season_position	was_home	points	team	value
index							
Erling Haaland	2023-08-11T19:00:00Z	Erling Haaland	20	1	10.194150	Man City	140
Harry Kane	2023-08-13T13:00:00Z	Harry Kane	9	1	9.730697	Spurs	125
Ivan Toney	2023-08-13T13:00:00Z	Ivan Toney	8	0	4.462744	Brentford	80
Aleksandar Mitrović	2023-08-12T14:00:00Z	Aleksandar Mitrović	17	1	3.465743	Fulham	75
Gabriel Fernando de Jesus	2023-08-12T12:00:00Z	Gabriel Fernando de Jesus	16	0	3.355168	Arsenal	80
Dominic Solanke	2023-08-12T14:00:00Z	Dominic Solanke	14	0	3.088639	Bournemouth	65
Ollie Watkins	2023-08-12T16:30:00Z	Ollie Watkins	4	1	2.984169	Aston Villa	80
Callum Wilson	2023-08-12T16:30:00Z	Callum Wilson	7	0	2.849204	Newcastle	80
Darwin Núñez Ribeiro	2023-08-13T15:30:00Z	Darwin Núñez Ribeiro	12	1	2.730184	Liverpool	75
Alexander Isak	2023-08-12T16:30:00Z	Alexander Isak	7	0	2.350178	Newcastle	75

	name	opponent	last_season_position	was_home	points	team	value
index							
Aaron Ramsdale	2023-08-12T12:00:00Z	Aaron Ramsdale	16	0	3.653835	Arsenal	50
Alisson Ramses Becker	2023-08-13T15:30:00Z	Alisson Ramses Becker	12	1	3.400355	Liverpool	55
David Raya Martin	2023-08-13T13:00:00Z	David Raya Martin	9	0	3.256289	Arsenal	50
José Malheiro de Sá	2023-08-14T19:00:00Z	José Malheiro de Sá	3	1	3.143887	Wolves	50
Emiliano Martínez Romero	2023-08-12T16:30:00Z	Emiliano Martínez Romero	4	1	2.944158	Aston Villa	50
Nick Pope	2023-08-12T16:30:00Z	Nick Pope	7	0	2.715660	Newcastle	55
Lukasz Fabianski	2023-08-12T14:00:00Z	Lukasz Fabianski	15	1	2.629725	West Ham	45
Ederson Santana de Moraes	2023-08-11T19:00:00Z	Ederson Santana de Moraes	20	1	2.539753	Man City	55
Bernd Leno	2023-08-12T14:00:00Z	Bernd Leno	17	1	2.145955	Fulham	45
Jordan Pickford	2023-08-12T14:00:00Z	Jordan Pickford	10	0	2.092276	Everton	45

3.1 Player Details

Get the predicted points for each player before Game Week from the Player Performance Prediction Model.



3.2 Sample Frontend 1

Get a Carousel guiding Managers to choose best Teams for Premier League season for each Gameweek (Demonstrated season 2022-23).

Game Week : 2

Captain(2x points) : Erling Haaland

Vice Captain : Rodrigo Moreno

Predicted_Total_Points : 85.73267326732673

Actual_Total_Points : 98

Total XI cost : 803

Predicted_Bench_Points : 12.673267326732674

Bench_value : 185

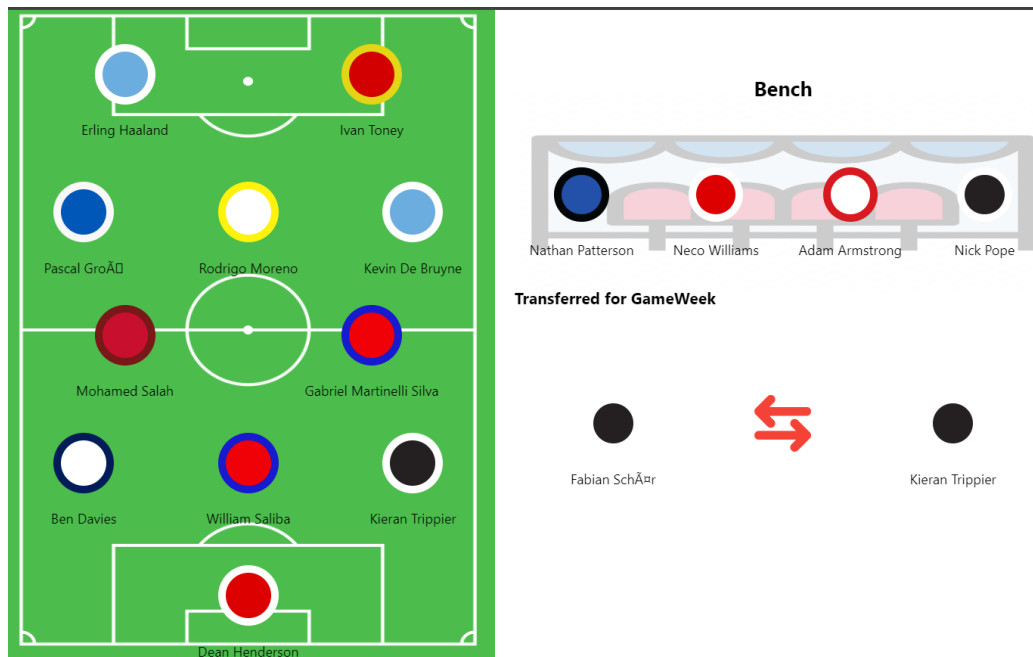
Remaining_budget : 16

Played GameChip: Bench Boost

GameChip Advantage Predicted: 13

53.2 Sample Frontend 2

For each GameWeek, get the Choice of Captain, ViceCaptain, Predicted Points, Actual Points (NA if match not played yet), Cost of XI, bench, Budget Remaining and GameChip to play for GameWeek and predicted Advantage if GameChip used.



3.3 Sample Frontend 3

Get the Squad to select for your Fantasy Team , the Starting XI (3-5-2), the Bench player and the transfer to make before the GameWeek.

7. RESULTS AND DISCUSSION

```
7.  {
8.      "GW": 6,
9.      "Starting_XI": {
10.          "GK": [
11.              {
12.                  "Name": "Nick Pope",
13.                  "Team": "Newcastle",
14.                  "TeamCSS": "Newcastle",
15.                  "ID": "Nick PopeNewcastleGK",
16.                  "Predicted_Points": 4.762376237623762,
17.                  "Cost": 50
18.              }
19.          ],
20.          "DEF": [
21.              {
22.                  "Name": "Ben Davies",
23.                  "Team": "Spurs",
24.                  "TeamCSS": "tottenham-hotspur",
25.                  "ID": "Ben DaviesSpursDEF",
26.                  "Predicted_Points": 2.633663366336634,
27.                  "Cost": 50
28.              },
29.              {
30.                  "Name": "William Saliba",
31.                  "Team": "Arsenal",
32.                  "TeamCSS": "arsenal",
33.                  "ID": "William SalibaArsenalDEF",
34.                  "Predicted_Points": 3.1386138613861387,
35.                  "Cost": 48
36.              },
37.              {
38.                  "Name": "Kieran Trippier",
39.                  "Team": "Newcastle",
40.                  "TeamCSS": "Newcastle",
41.                  "ID": "Kieran TrippierNewcastleDEF",
42.                  "Predicted_Points": 5.673267326732673,
43.                  "Cost": 51
44.              }
45.          ],
46.          "MID": [
47.              {
48.                  "Name": "Alexis Mac Allister",
49.                  "Team": "Brighton",
50.                  "TeamCSS": "brighton",
51.                  "ID": "Alexis Mac AllisterBrightonMID",
52.                  "Predicted_Points": 4.316831683168317,
53.                  "Cost": 55
54.              },
```

```

55.         {
56.             "Name": "Bernardo Veiga de Carvalho e Silva",
57.             "Team": "Man City",
58.             "TeamCSS": "manchester-city",
59.             "ID": "Bernardo Veiga de Carvalho e SilvaMan
CityMID",
60.             "Predicted_Points": 3.108910891089109,
61.             "Cost": 70
62.         },
63.         {
64.             "Name": "Kevin De Bruyne",
65.             "Team": "Man City",
66.             "TeamCSS": "manchester-city",
67.             "ID": "Kevin De BruyneMan CityMID",
68.             "Predicted_Points": 5.564356435643564,
69.             "Cost": 122
70.         },
71.         {
72.             "Name": "Leandro Trossard",
73.             "Team": "Brighton",
74.             "TeamCSS": "brighton",
75.             "ID": "Leandro TrossardBrightonMID",
76.             "Predicted_Points": 6.346534653465347,
77.             "Cost": 65
78.         },
79.         {
80.             "Name": "Gabriel Martinelli Silva",
81.             "Team": "Arsenal",
82.             "TeamCSS": "arsenal",
83.             "ID": "Gabriel Martinelli SilvaArsenalMID",
84.             "Predicted_Points": 3.108910891089109,
85.             "Cost": 65
86.         }
87.     ],
88.     "FWD": [
89.         {
90.             "Name": "Erling Haaland",
91.             "Team": "Man City",
92.             "TeamCSS": "manchester-city",
93.             "ID": "Erling HaalandMan CityFWD",
94.             "Predicted_Points": 7.1683168316831685,
95.             "Cost": 119
96.         },
97.         {
98.             "Name": "Roberto Firmino",
99.             "Team": "Liverpool",
100.             "TeamCSS": "liverpool",
101.             "ID": "Roberto FirminoLiverpoolFWD",
102.             "Predicted_Points": 2.6831683168316833,
103.             "Cost": 81

```

```

104.         }
105.     ]
106. },
107.     "Captain": "Erling Haaland",
108.     "Vice_Captain": "Leandro Trossard",
109.     "Predicted_Total_Points": 55.67326732673268,
110.     "Actual_Total_Points": 77,
111.     "XI_Cost": 776,
112.     "Bench": [
113.         {
114.             "Name": "Nathan Patterson",
115.             "Position": "DEF",
116.             "Team": "Everton",
117.             "TeamCSS": "everton",
118.             "ID": "Nathan PattersonEvertonDEF",
119.             "Predicted_Points": 2.405940594059406,
120.             "Cost": 40
121.         },
122.         {
123.             "Name": "Neco Williams",
124.             "Position": "DEF",
125.             "Team": "Nott'm Forest",
126.             "TeamCSS": "not-forest",
127.             "ID": "Neco WilliamsNott'm ForestDEF",
128.             "Predicted_Points": 1.900990099009901,
129.             "Cost": 41
130.         },
131.         {
132.             "Name": "Adam Armstrong",
133.             "Position": "FWD",
134.             "Team": "Southampton",
135.             "TeamCSS": "southampton",
136.             "ID": "Adam ArmstrongSouthamptonFWD",
137.             "Predicted_Points": 1.4752475247524752,
138.             "Cost": 55
139.         },
140.         {
141.             "Name": "Dean Henderson",
142.             "Position": "GK",
143.             "Team": "Nott'm Forest",
144.             "TeamCSS": "not-forest",
145.             "ID": "Dean HendersonNott'm ForestGK",
146.             "Predicted_Points": 2.128712871287129,
147.             "Cost": 47
148.         }
149.     ],
150.     "Predicted_Bench_Points": 7.910891089108911,
151.     "Bench_value": 183,
152.     "Remaining_budget": 70,
153.     "Transferred_Out": {

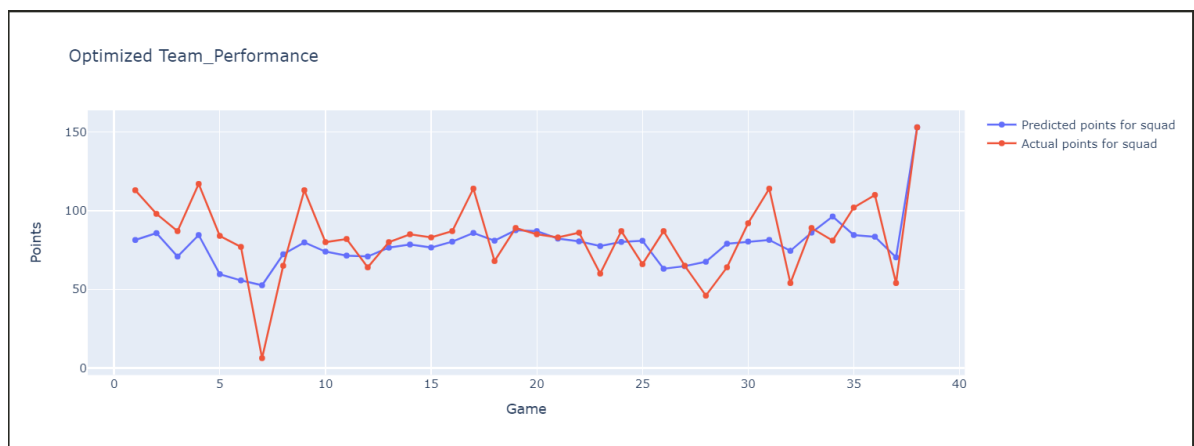
```

```

154.         "Name": "Mohamed Salah",
155.         "Position": "MID",
156.         "Team": "Liverpool",
157.         "TeamCSS": "liverpool",
158.         "ID": "Mohamed SalahLiverpoolMID",
159.         "Predicted_Points": 2.712871287128713,
160.         "Cost": 130
161.     },
162.     "Transferred_In": {
163.         "Name": "Leandro Trossard",
164.         "Position": "MID",
165.         "Team": "Brighton",
166.         "TeamCSS": "brighton",
167.         "ID": "Leandro TrossardBrightonMID",
168.         "Predicted_Points": 4.237623762376238,
169.         "Cost": 68
170.     }
171. }

```

Data calculated for GW 7



3.5 Predicted points comparison

Predicted Points vs Actual Points for season 2022-23

```

prediction_error_rate = (sum(actualPoints)-sum(predictedPoints))/38
prediction_error_rate

```

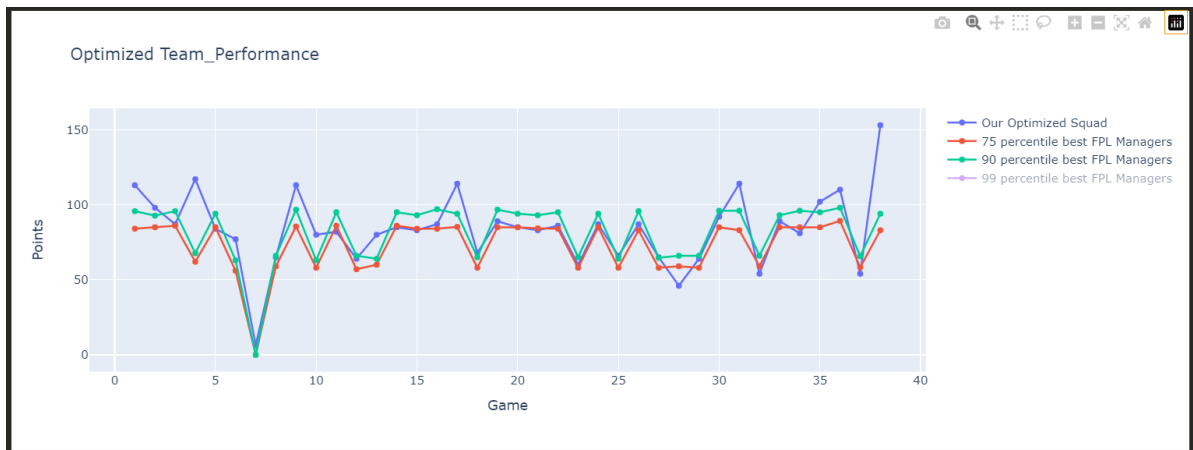
✓ 0.0s

4.54448133997855

Mean Difference between Actual and Predicted points.

```
int(sum(actualPoints))  
✓ 0.0s  
3170
```

Total Points at the end of season by our Team



3.6 FPL points comparison against other Managers

Our Optimized Team performance against data of 205 top managers from VIT official FPL league.

8. SUMMARY

Our project represents a significant advancement in Fantasy Premier League (FPL) team management, leveraging cutting-edge technologies to provide FPL managers with unparalleled insights and predictive analytics. By combining data analytics, machine learning, and artificial intelligence, we have developed sophisticated models such as SARIMAX and Light GBM for precise time series forecasting and player performance prediction. These models have demonstrated exceptional accuracy, outperforming over 90% of all teams, as indicated by evaluations from the Premier League website.

To ensure the robustness of our modeling approach, we conducted Discrete Fourier Transform testing, allowing us to analyze seasonal patterns in the time series data and refine our models accordingly. Moreover, our data assimilation process involves integrating historical time series data and other relevant factors, enriching our models with comprehensive information for training and analysis.

In addition to modeling, we conduct sentiment analysis on news headlines using advanced natural language processing techniques, including TextBlob and BERT models. This analysis provides valuable insights into player sentiment within the FPL community, enabling managers to factor in external perceptions when making strategic decisions.

On the frontend side, our module, developed with React.js and AliceCarousel, offers an intuitive and visually engaging interface for FPL managers. The dynamic presentation of team data in a carousel format enhances user interaction and facilitates comprehensive data visualization, empowering managers to assess their team compositions effectively. Furthermore, we employ dynamic programming techniques to optimize squad selection, considering budget constraints and positional limits. This approach ensures that managers can maximize their team's potential points while adhering to strategic constraints, ultimately enhancing their performance within the league.

Overall, our project represents a transformative step forward in FPL team management, revolutionizing the FPL experience for managers. By integrating advanced technologies and adhering to rigorous modeling and analysis methodologies, we provide managers with the tools they need to make informed decisions, optimize team performance, and achieve success in the league.

9. REFERENCES

1. Rajesh, V., Arjun, P., Jagtap, K. R., Suneera, C. M., & Prakash, J. (2022, June). Player Recommendation System for Fantasy Premier League using Machine Learning. In *2022 19th International Joint Conference on Computer Science and Software Engineering (JCSSE)* (pp. 1-6). IEEE.
2. Apostolou, K., & Tjortjis, C. (2019, July). Sports Analytics algorithms for performance prediction. In *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)* (pp. 1-4). IEEE.
3. Tuyls, K., Omidshafiei, S., Muller, P., Wang, Z., Connor, J., Hennes, D., ... & Hassabis, D. (2021). Game Plan: What AI can do for Football, and What Football can do for AI. *Journal of Artificial Intelligence Research*, 71, 41-88.
4. Bonello, N., Beel, J., Lawless, S., & Debattista, J. (2019). Multi-stream data analytics for enhanced performance prediction in fantasy football.
5. Yiğit, A. T., Samak, B., & Kaya, T. (2020). Football player value assessment using machine learning techniques. In *Intelligent and Fuzzy Techniques in Big Data Analytics and Decision Making: Proceedings of the INFUS 2019 Conference, Istanbul, Turkey, July 23-25, 2019* (pp. 289-297). Springer International Publishing.
6. Pokharel, P., Timalsina, A., Panday, S. and Acharya, B., 2022. Fantasy Premier League-Performance Prediction.
7. Bhatt, S., Chen, K., Shalin, V.L., Sheth, A.P. and Minnery, B., 2019, July. Who should be the captain this week? Leveraging inferred diversity-enhanced crowd wisdom for a fantasy premier league captain prediction. In *Proceedings of the international AAAI conference on Web and Social Media* (Vol. 13, pp. 103-113).
8. Gupta, A., 2019. Time series modeling for dream team in fantasy premier league.
9. Silva, G.H.D.S., 2021. Football players performance analysis and formal/informal media: *Sentiment analysis and semantic similarity (Master's thesis)*.
10. Padmanabhan, K.A., TF-Pundit: A Real-time Football Pundit based on Twitter.

APPENDIX A – SAMPLE CODE

Data Engineering (sample)

```
import numpy as np
import pandas as pd
from scipy import stats

def split_test(data, gameweek):
    # print(data["GW"].value_counts())
    data_gw = data[data["GW"] == gameweek]
    data_other_gw = data[~(data["GW"] == gameweek)]
    return data_gw, data_other_gw

def check_win(df):
    list_win = []
    for index in df.index:
        result = (df["team_a_score"] - df["team_h_score"]).loc[index]
        is_home = df["was_home"].loc[index]
        if result == 0:
            list_win.append(1)
        elif result > 0 and is_home == True:
            list_win.append(0)
        elif result < 0 and is_home == True:
            list_win.append(3)
        elif result > 0 and is_home == False:
            list_win.append(3)
        elif result < 0 and is_home == False:
            list_win.append(0)
        else:
            list_win.append(-1)
    return list_win

def get_2020_21_season_pos(club):
    """get the position of the club in the 2020-21 season"""
    if club == "Man City":
        return 1
    elif club == "Man Utd":
        return 2
    elif club == "Liverpool":
        return 3
    elif club == "Chelsea":
        return 4
    elif club == "Leicester":
        return 5
    elif club == "West Ham":
        return 6
    elif club == "Spurs":
        return 7
    elif club == "Arsenal":
```

```

        return 8
    elif club == "Leeds":
        return 9
    elif club == "Everton":
        return 10
    elif club == "Aston Villa":
        return 11
    elif club == "Newcastle":
        return 12
    elif club == "Wolves":
        return 13
    elif club == "Crystal Palace":
        return 14
    elif club == "Southampton":
        return 15
    elif club == "Brighton":
        return 16
    elif club == "Burnley":
        return 17
    else:
        return 20

def get_2019_20_season_pos(club):
    """get the position of the club in the 2019-20 season"""
    if club == "Liverpool":
        return 1
    elif club == "Man City":
        return 2
    elif club == "Man Utd":
        return 3
    elif club == "Chelsea":
        return 4
    elif club == "Leicester":
        return 5
    elif club == "Spurs":
        return 6
    elif club == "Wolves":
        return 7
    elif club == "Arsenal":
        return 8
    elif club == "Sheffield Utd":
        return 9
    elif club == "Burnley":
        return 10
    elif club == "Southampton":
        return 11
    elif club == "Everton":
        return 12
    elif club == "Newcastle":
        return 13
    elif club == "Crystal Palace":
        return 14
    elif club == "Brighton":

```

```

        return 15
    elif club == "West Ham":
        return 16
    elif club == "Aston Villa":
        return 17
    else:
        return 20

def get_2021_22_season_pos(club):
    if club == "Man City":
        return 1
    elif club == "Liverpool":
        return 2
    elif club == "Chelsea":
        return 3
    elif club == "Spurs":
        return 4
    elif club == "Arsenal":
        return 5
    elif club == "Man Utd":
        return 6
    elif club == "West Ham":
        return 7
    elif club == "Leicester":
        return 8
    elif club == "Brighton":
        return 9
    elif club == "Wolves":
        return 10
    elif club == "Newcastle":
        return 11
    elif club == "Crystal Palace":
        return 12
    elif club == "Brentford":
        return 13
    elif club == "Aston Villa":
        return 14
    elif club == "Southampton":
        return 15
    elif club == "Everton":
        return 16
    elif club == "Leeds":
        return 17
    else:
        return 20

def get_2022_23_season_pos(club):
    if club == "Man City":
        return 1
    elif club == "Arsenal":
        return 2
    elif club == "Man Utd":

```

```

        return 3
    elif club == "Newcastle":
        return 4
    elif club == "Liverpool":
        return 5
    elif club == "Brighton":
        return 6
    elif club == "Aston Villa":
        return 7
    elif club == "Spurs":
        return 8
    elif club == "Brentford":
        return 9
    elif club == "Fulham":
        return 10
    elif club == "Crystal Palace":
        return 11
    elif club == "Chelsea":
        return 12
    elif club == "Wolves":
        return 13
    elif club == "West Ham":
        return 14
    elif club == "Bournemouth":
        return 15
    elif club == "Nott'm Forest":
        return 16
    elif club == "Everton":
        return 17
    else:
        return 20

def get_last_season_pos(year):
    """get the function to get the last season position of a team at any ye
ar"""
    if year == "2020-21":
        return get_2019_20_season_pos
    elif year == "2021-22":
        return get_2020_21_season_pos
    elif year == "2022-23":
        return get_2021_22_season_pos
    elif year == "2023-24":
        return get_2022_23_season_pos

def remove_neg(val):
    if val > 0:
        return val
    else:
        return -val

def deque_and_queue(stats, value):

```

```

    # if -1 in stats:
    #     return stats
    # deque
    stats = stats[1:]
    stats.append(value)
    return stats

def get_all_stats(data, stat, name):
    '''All stats for a player'''
    name_df = data[data["name"] == name]
    seasons = ["2020-21", "2021-22", "2022-23"]
    name_df_dict = {}
    for season in seasons:
        name_df_season = name_df[name_df["season"] == season]
        list_stats = []
        stats_x = []
        for value in name_df_season[stat]:
            list_stats.append(np.array(stats_x))
            stats_x.append(value)

        name_df_season[f"all {stat}"] = list_stats
        name_df_dict[season] = name_df_season
    return pd.concat(name_df_dict.values())

def get_last_stats(data, stat, name, no_last_stats=short_term_stats):
    '''Get last n week stats for specific player'''
    name_df = data[data["name"] == name]
    seasons = ["2020-21", "2021-22", "2022-23"]
    name_df_dict = {}
    for season in seasons:
        name_df_season = name_df[name_df["season"] == season]
        list_stats = []
        stats_x = []
        for value in name_df_season[stat]:
            if len(stats_x) < no_last_stats:
                list_stats.append(np.array(stats_x))
                stats_x.append(value)
            else:
                list_stats.append(np.array(stats_x))
                stats_x = deque_and_queue(stats_x, value)
        name_df_season[f"last {no_last_stats} {stat}"] = list_stats
        name_df_dict[season] = name_df_season
    return pd.concat(name_df_dict.values())

def get_all_players_last_stats(data, stat, no_last_stats=short_term_stats):
    '''Get last n week stats for all players'''
    players_df = []
    for player in data["name"].unique():
        data_player = get_last_stats(data, stat, player, no_last_stats)
        players_df.append(data_player)

```

```

# print(pd.concat(players_df))
return pd.concat(players_df)

def get_all_players_all_stats(data, stat):
    '''Get all stats for all players'''
    players_df = []
    for player in data["name"].unique():
        data_player = get_all_stats(data, stat, player)
        players_df.append(data_player)
    # print(pd.concat(players_df))
    return pd.concat(players_df)

def get_last_stats_test(data, stat, name, no_last_stats=short_term_stats):
    '''Get last n week stats for specific players'''
    name_df = data[data["name"] == name]
    list_stats = []
    stats_x = []
    for value in name_df[stat]:
        if len(stats_x) < no_last_stats:
            list_stats.append(np.array(stats_x))
            stats_x.append(value)
        else:
            list_stats.append(np.array(stats_x))
            stats_x = deque_and_queue(stats_x, value)
    name_df[f"last {no_last_stats} {stat}"] = list_stats
    return name_df

def get_players_last_stats_test(data, stat, no_last_stats=short_term_stats):
    '''Get last n week stats for all players'''
    players_df = []
    for player in data["name"].unique():
        data_player = get_last_stats_test(data, stat, player, no_last_stats)
    players_df.append(data_player)
    # print(pd.concat(players_df))
    return pd.concat(players_df)

def get_all_stats_test(data, stat, name):
    '''Get all stats for specific player'''
    name_df = data[data["name"] == name]
    list_stats = []
    stats_x = []
    for value in name_df[stat]:
        list_stats.append(np.array(stats_x))
        stats_x.append(value)
    name_df[f"all {stat}"] = list_stats
    return name_df

def get_players_all_stats_test(data, stat):
    '''Get all stats for all player'''
    players_df = []

```

```

for player in data["name"].unique():
    data_player = get_all_stats_test(data, stat, player)
    players_df.append(data_player)
# print(pd.concat(players_df))
return pd.concat(players_df)

def convert_minutes(val):
    """CONVERTS MINUTES TO A CATEGORICAL OUTPUT"""
    if val > 10:
        return 1
    else:
        return 0

def find_mode(vals):
    """find the mode of vals"""
    try:
        if -1 in vals:
            return -1
        return stats.mode(vals)[0][0]
    except IndexError:
        return np.nan

def find_mean(vals):
    """find the mean of vals"""
    try:
        if -1 in vals:
            return -1
        return np.mean(vals)
    except:
        return np.nan

def find_max(vals):
    """find the maximum of vals"""
    try:
        if -1 in vals:
            return -1
        return np.max(vals)
    except:
        return np.nan

def find_std(vals):
    """find the standard deviation of vals"""
    try:
        if -1 in vals:
            return -1
        return np.std(vals)
    except:
        return np.nan

```

```
def find_value_count(vals, to_count):
    """find the number of times to_count appears in vals"""
    try:
        if -1 in vals:
            return -1
        values, count = np.unique(vals, return_counts=True)
        index = np.where(values == to_count)[0][0]
        return count[index]

    except:
        return -2

....
```

Model Training

```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.ensemble import (
    RandomForestClassifier,
    RandomForestRegressor,
    GradientBoostingRegressor,
)
from sklearn.preprocessing import StandardScaler, RobustScaler, MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import (
    mean_squared_error,
    mean_absolute_error,
    confusion_matrix,
    accuracy_score,
    f1_score,
)
from lightgbm import LGBMRegressor, LGBMClassifier
from catboost import CatBoostClassifier, CatBoostRegressor
from sklearn.model_selection import StratifiedKFold, KFold

x, val, y, y_val = train_test_split(
    train.drop(leak_columns, axis=1),
    target["total_points"],
    test_size=0.1,
    random_state=0,
)

from sklearn.model_selection import KFold

folds=KFold(n_splits=8,shuffle=True,random_state=0)
```



```

predictions_df=pd.DataFrame()

#list to save the mean absolute errors from validating on each folds
rmse_val=[]
rmse_X=[]

model=LGBMRegressor(**{'colsample_bytree': 0.4199299182268318,
'learning_rate': 0.0032874466037521254, 'max_depth': 9, 'min_split_gain':
0.5685369160138952, 'num_leaves': 99, 'reg_alpha': 0.5621526419488447,
'reg_lambda': 0, 'subsample': 0.6534153111773866}, verbose=-
50,random_state=0,early_stopping_rounds=200,n_estimators=10000)

#train model, make predictions and check the validation accuracy on each
fold
for i,(train_index,test_index) in
enumerate(folds.split(train.drop(leak_columns,
axis=1),target["total_points"])):
    train_fold=train.drop(leak_columns, axis=1).iloc[train_index]
    val_fold=train.drop(leak_columns, axis=1).iloc[test_index]
    y_fold=target["total_points"].iloc[train_index]
    y_val_fold=target["total_points"].iloc[test_index]

    model.fit(train_fold,y_fold,eval_set=[(val_fold,y_val_fold)])
    print(i+1)
    prediction=model.predict(test.drop(leak_columns, axis=1))
    predictions_df[i]=prediction
    rmse_val.append(mean_squared_error(model.predict(val_fold),y_val_fold,squ
ared=False))
    rmse_X.append(mean_squared_error(model.predict(train_fold),y_fold,squared
=False))
print(rmse_val)
print(rmse_X)

print(np.mean(rmse_val))
print(np.mean(rmse_X))

```

Sentiment Analysis

```

import requests
from bs4 import BeautifulSoup
# Last name
last_name = "son" # Replace with the desired last name

# URL with placeholder for last name
URL = 'https://www.bbc.com/search?q=premier%20league%20{'

```

```

# Replace the placeholder with the last name
URL = URL.format(last_name)

# Send a GET request to the URL
response = requests.get(URL)

# Print the response
print(response)
headlines = soup.find_all(class_='sc-4fedabc7-3 bvDsJq')

# Find all card descriptions
descriptions = soup.find_all(class_='sc-ae29827d-0 cNPpME')

# Print headlines and descriptions
for headline, description in zip(headlines, descriptions):
    print("Headline:", headline.text)
    print("Description:", description.text)
from textblob import TextBlob
# Loop through headlines and descriptions
for headline, description in zip(headlines, descriptions):
    # Combine headline and description text
    text = headline.text + ' ' + description.text

    # Perform sentiment analysis
    analysis = TextBlob(text)

    # Print sentiment polarity and subjectivity
    print("Text:", text)
    print("Sentiment Polarity:", analysis.sentiment.polarity)
    print("Sentiment Subjectivity:", analysis.sentiment.subjectivity)
    print("\n")
from transformers import BertTokenizer, BertForSequenceClassification
import torch

# Load pre-trained BERT model and tokenizer
model_name = 'bert-base-uncased'
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name)
# Initialize lists to store sentiment polarities
sentiment_polarities = []

# Perform sentiment analysis for each data point
for headline, description in zip(headlines, descriptions):
    # Combine headline and description
    text = headline.text + ' ' + description.text

    # Tokenize the text
    inputs = tokenizer(text, return_tensors='pt', max_length=512,
truncation=True)

```

```

# Perform inference
outputs = model(**inputs)

# Extract predicted sentiment
predicted_class = torch.argmax(outputs.logits).item()

# Map predicted class to sentiment label
sentiments = ['negative', 'neutral', 'positive']
predicted_sentiment = sentiments[predicted_class]

# Print results
#print("Headline:", headline)
#print("Description:", description)
print("text:", text)
print("Predicted Sentiment:", predicted_sentiment)
print()

# Calculate sentiment polarity
if predicted_sentiment == 'positive':
    sentiment_polarity = 1
elif predicted_sentiment == 'negative':
    sentiment_polarity = -1
else:
    sentiment_polarity = 0

# Append sentiment polarity to the list
sentiment_polarities.append(sentiment_polarity)

# Calculate aggregated sentiment polarity
aggregated_sentiment = sum(sentiment_polarities) /
len(sentiment_polarities)
print("Aggregated Sentiment Polarity:", aggregated_sentiment)

```

Optimization

```

def makeChanges(gw, initDf):
    takenSquad = [0]*len(teams)
    initDf = initDf[:]
    # gw = 2
    saved_transfer = 0
    data = giveData(gw)
    initData = pruneData(data)
    prunedDf = pd.DataFrame(initData,
columns=['ignore','name','pos','team','id', 'points', 'value',
'value_inTeam','efficiency'])
    # effData = prunedDf.sort_values(by="efficiency",
ascending=False).drop(columns=['ignore','efficiency','value_inTeam']).values.
tolist()

```

```

    optDf = prunedDf.sort_values(by="points",
ascending=False).drop(columns=['ignore', 'efficiency', 'value_inTeam']).head(40
).reset_index()
    rest_best = []
    new_XI = []
    curr_XI = initDf[1]
    pos_XI = []
    points_XI = []
    rank_XI = []
    cost_XI = []
    pos_bench = []
    points_bench = []
    cost_bench = []
    rank_bench = []
    curr_bench = initDf[7]
    curr_budget = initDf[-1]
    transfer_limit = 1 + saved_transfer
    for i in curr_XI:
        try:
            points_XI.append(prunedDf[prunedDf["id"]==i[3]]['points'].values[
0]))
        except:
            # print(i)
            points_XI.append(0)
        try:
            cost_XI.append(prunedDf[prunedDf["id"]==i[3]]['value'].values[0])
        except:
            cost_XI.append(finalDf[finalDf["id"]==i[3]]['value'].values[0])
        try:
            pos_XI.append(prunedDf[prunedDf["id"]==i[3]]['pos'].values[0])
        except:
            pos_XI.append(finalDf[finalDf["id"]==i[3]]['position'].values[0])
        takenSquad[teamsDic[i[2]]]+=1
        try:
            rank_XI.append(optDf[optDf["id"]==i[3]].index.tolist()[0])
        except Exception as e:
            # print(i)
            rank_XI.append(30)
        optDf = optDf[optDf["id"]!=i[3]]

    for b in curr_bench:
        try:
            points_bench.append(prunedDf[prunedDf["id"]==b[3]]['points'].valu
es[0]))
        except:
            # print(b)
            points_bench.append(0)
        try:
            cost_bench.append(prunedDf[prunedDf["id"]==b[3]]['value'].values[
0]))

```

```

        except:
            cost_bench.append(finalDf[finalDf["id"]==b[3]]['value'].values[0])
    )

    try:
        pos_bench.append(prunedDf[prunedDf["id"]==b[3]]['pos'].values[0])
    except:
        pos_bench.append(finalDf[finalDf["id"]==b[3]]['position'].values[0])

0])

takenSquad[teamsDic[b[2]]]+=1
try:
    rank_bench.append(optDf[optDf["id"]==b[3]].index.tolist()[0])
except Exception as e:
    # print(i)
    rank_bench.append(30)
optDf = optDf[optDf["id"]!=b[3]]

# print(curr_XI)
# print(curr_bench)
# print(curr_budget)
# print("-----")
# print(takenSquad)
# print([points_XI,cost_XI,pos_XI,rank_XI])
# print([points_bench,cost_bench,pos_bench,rank_bench])
# optDf
# Optimize XI vs bench
for i in range(len(curr_bench)):
    for j in range(len(curr_XI)):
        if pos_bench[i] == pos_XI[j] and points_bench[i]>points_XI[j]:
            # print(curr_XI[j],"->")
            # print(curr_bench[i],"<-")
            temp = curr_bench[i]
            curr_bench[i] = curr_XI[j]
            curr_XI[j] = temp
            temp = points_bench[i]
            points_bench[i] = points_XI[j]
            points_XI[j] = temp
            temp = cost_bench[i]
            cost_bench[i] = cost_XI[j]
            cost_XI[j] = temp
            temp = rank_bench[i]
            rank_bench[i] = rank_XI[j]
            rank_XI[j] = temp

# print(curr_XI)
# print(curr_bench)
# print(curr_budget)
# print("-----")
# print(takenSquad)
# print([points_XI,cost_XI,pos_XI,rank_XI])
# print([points_bench,cost_bench,pos_bench,rank_bench])

```

```

newDf = optDf.drop(columns=['index'])
newDf['rank'] = newDf.index.tolist()
newDf = newDf.reset_index().drop(columns=['index'])
newDf

# [[players from XI],[transfers from optDf(newDf)], points profit, new
budget
feasible_transfers = []
points_profit = []
new_budget = []

for i in range(len(curr_XI)):
    try:
        tempDf =
newDf[newDf["pos"]==pos_XI[i]][newDf["rank"]<=rank_XI[i]]
        for t in range(len(tempDf)):
            transfer = tempDf.iloc[t]
            if takenSquad[teamsDic[transfer['team']]]<3 and
(curr_budget+cost_XI[i]-transfer['value'])>=0:
                feasible_transfers.append([i,transfer['rank']])
                points_profit.append(transfer['points']-points_XI[i])
                new_budget.append(curr_budget+cost_XI[i]-
transfer['value'])

        except Exception as e:
            pass

    # transferDf =
pd.DataFrame([feasible_transfers,points_profit,new_budget])
    # selected_transfer = transferDf.iloc[0]

    # print(feasible_transfers)
    # print(points_profit)
    # print(new_budget)

    combined_list = list(zip(feasible_transfers, points_profit, new_budget))
    sorted_data = sorted(combined_list, key=lambda x: -x[1])
    sorted_transfers, sorted_points_profit, sorted_new_budget =
zip(*sorted_data)
    # print(sorted_transfers[0],sorted_points_profit[0],
sorted_new_budget[0])
    transfer_out = curr_XI[sorted_transfers[0][0]]
    # print(transfer_out,"==>")
    # # print(curr_XI[sorted_transfers[0][0]], "==>")
    # print(points_XI[sorted_transfers[0][0]])
    transfer_in =
newDf[newDf["rank"]==sorted_transfers[0][1]].iloc[0].values.tolist()[:-1]
    # print(transfer_in,"<==")
    # print(newDf[newDf["rank"]==sorted_transfers[0][1]].iloc[0]['points'])

    new_XI = curr_XI[:]

```

```

# prunedDf
for i in range(len(new_XI)):
    new_XI[i][4] = points_XI[i]
    new_XI[i][5] = cost_XI[i]
    if new_XI[i][3] == transfer_out[3]:
        transfer_out[4] = points_XI[sorted_transfers[0][0]]
        transfer_out[5] = cost_XI[sorted_transfers[0][0]]
        new_XI[i] = transfer_in
        # points_XI[sorted_transfers[0][0]] = transfer_in[4]
        # cost_XI[sorted_transfers[0][0]] = transfer_in[5]
        # rank_XI[sorted_transfers[0][0]] = sorted_transfers[0][1]
        takenTeam[teamsDic[transfer_out[2]]]-=1
        takenTeam[teamsDic[transfer_in[2]]]+=1
# temp_XI = new_XI
# new_XI = curr_XI
# curr_XI = temp_XI
new_bench = curr_bench[:]
for i in range(len(new_bench)):
    new_bench[i][4] = points_bench[i]
    new_bench[i][5] = cost_bench[i]

# print(sorted_new_budget[0])
# print(new_XI)
# print(new_bench)
# print(transfer_out)
# print(transfer_in)

starters = new_XI[:]
predictedpoints = []
actualpoints = []
starters_value = 0
captain = sorted(starters, key=lambda x: -x[4])[0]
vicecaptain = sorted(starters, key=lambda x: -x[4])[1]
# for s in starters:
#     predicted_points+=s[4]
# predicted_points+=captain[4]
for s in starters:
    predictedpoints.append(s[4])
    try:
        starters_value+=prunedDf[prunedDf["id"]==s[3]].iloc[0]['value']
    except:
        starters_value+=finalDf[finalDf["id"]==s[3]]['value'].values[0]
    try:
        actualpoints.append(finalDf[finalDf["id"] == s[3]][finalDf["GW"]
== gw]["total_points"].tolist()[0])
    except Exception as e:
        actualpoints.append(0)
        # print(s[3])
predPoints = sum(predictedpoints)+captain[4]
try:

```

```

        realPoints = sum(actualpoints)+finalDf[finalDf["id"] ==
captain[3]][finalDf["GW"] == gw]["total_points"].tolist()[0]
    except Exception as e:
        # print(finalDf[finalDf["id"] == captain[3]][finalDf["GW"] ==
gw]["total_points"].tolist()[0])
        # print(actualpoints)
        realPoints = sum(actualpoints) + captain[4]

predbenchpoints = []
bench_value = 0
for b in new_bench:
    predbenchpoints.append(b[4])
    try:
        bench_value+=prunedDf[prunedDf["id"]==b[3]].iloc[0]['value']
    except:
        bench_value+=finalDf[finalDf["id"]==b[3]]['value'].values[0]

res = [gw]
res.append(starters[:])
res.append(captain[:])
res.append(vicecaptain[:])
res.append(predPoints)
res.append(realPoints)
res.append(starters_value)
res.append(new_bench[:])
res.append(sum(predbenchpoints))
res.append(bench_value)
res.append(sorted_new_budget[0])
# print(res)

transfer_info = [gw]
transfer_info.append([transfer_out,transfer_in])
# print(transfer_info)

return res, transfer_info
def giveBench(data,prunedDf):
    benchDf = prunedDf.sort_values(by=['value', 'points', 'pos'],
ascending=[True,False,True])
    try:
        bench =
benchDf[benchDf["pos"]=="DEF"][benchDf["efficiency"]>0.5].iloc[0:2,1:7].value
s.tolist()+[benchDf[benchDf["pos"]=="FWD"][benchDf["efficiency"]>0.5].iloc[0,
1:7].values.tolist()+[benchDf[benchDf["pos"]=="GK"][benchDf["efficiency"]>0.
5].iloc[0,1:7].values.tolist()]
    except Exception as e:
        bench = curr_bench
        bench_pts = 0
        bench_value = 0
        for p in bench:

```



```

        # print(p)
        try:
            data.remove(p)
        except Exception as e:
            pass
        # print(p)
        takenTeam[teamsDic[p[2]]] += 1
        bench_pts += p[4]
        bench_value += p[5]
    # print(bench)
    # print(bench_pts)
    # print(bench_value)
    return [bench_pts, bench, bench_value]

def giveSquad(data, budget):
    n = len(data)

    dp = {}
    teamSquad = []
    team = []

    def
f(idx, budget, maxGoalKeeper, maxDefender, maxMidFielder, maxForward):

        if(maxGoalKeeper == 0 and maxDefender == 0 and maxMidFielder == 0 and
maxForward == 0):
            return [0, list(team)]

        if(idx == n):
            if(maxGoalKeeper != 0 or maxDefender != 0 or maxMidFielder != 0
or maxForward != 0):
                return [-math.inf, []]
            return [0, list(team)]

        if((idx, budget, maxGoalKeeper, maxDefender, maxMidFielder, maxForward, tuple(
le(takenTeam)) in dp):
            return
dp[(idx, budget, maxGoalKeeper, maxDefender, maxMidFielder, maxForward, tuple(taken
Team))]

        name, pos, teamPlayer, id, val, wt = data[idx]
        pick = -math.inf
        if(budget >= wt and takenTeam[teamsDic[teamPlayer]] < teamLimit and
val!=0):
            if(pos == "DEF"):
                if(maxDefender > 0):
                    takenTeam[teamsDic[teamPlayer]] += 1

```

```

        team.append(idx)
        [pick, pickedTeam] = f(idx + 1, budget -
wt, maxGoalKeeper, maxDefender - 1, maxMidFielder, maxForward)
        team.pop()
        pick += val
        takenTeam[teamsDic[teamPlayer]] -= 1
    elif(pos == "MID"):
        if(maxMidFielder > 0):
            takenTeam[teamsDic[teamPlayer]] += 1
            team.append(idx)
            [pick, pickedTeam] = f(idx + 1, budget -
wt, maxGoalKeeper, maxDefender, maxMidFielder - 1, maxForward)
            team.pop()
            pick += val
            takenTeam[teamsDic[teamPlayer]] -= 1
    elif(pos == "FWD"):
        if(maxForward > 0):
            takenTeam[teamsDic[teamPlayer]] += 1
            team.append(idx)
            [pick, pickedTeam] = f(idx + 1, budget -
wt, maxGoalKeeper, maxDefender, maxMidFielder, maxForward - 1)
            team.pop()
            pick += val
            takenTeam[teamsDic[teamPlayer]] -= 1
    else:
        if(maxGoalKeeper > 0):
            takenTeam[teamsDic[teamPlayer]] += 1
            team.append(idx)
            [pick, pickedTeam] = f(idx + 1, budget - wt, maxGoalKeeper
- 1, maxDefender, maxMidFielder, maxForward)
            team.pop()
            pick += val
            takenTeam[teamsDic[teamPlayer]] -= 1
        [notPick, notpickedTeam] = f(idx +
1, budget, maxGoalKeeper, maxDefender, maxMidFielder, maxForward)

        if(pick > notPick):
            dp[(idx, budget, maxGoalKeeper, maxDefender, maxMidFielder, maxForward
, tuple(takenTeam))] = [pick, pickedTeam]
            return [pick, list(pickedTeam)]
            dp[(idx, budget, maxGoalKeeper, maxDefender, maxMidFielder, maxForward, tup
le(takenTeam))] = [notPick, notpickedTeam]
            return [notPick, list(notpickedTeam)]

    return f(0, budget, 1, 3, 5, 2)
...

```

Frontend

```

import React from 'react';
import AliceCarousel from 'react-alice-carousel';
import 'react-alice-carousel/lib/alice-carousel.css';
import { useEffect, useState } from 'react';
import './Carousel.css'
import myImage from '../Football_field.jpg';
import bench from '../bench.jpg';
import swap from '../swap.jpg';
import Button from '@mui/material/Button';
import Tooltip from '@mui/material/Tooltip';

const handleDragStart = (e) => e.preventDefault();

const Carousel = ({data}) => {
  const items = data.data.map((item,idx)=>(
    <div className='carouselItem'>
      <div className='left'>
        <h1>Game Week - {item.GW}</h1>
        <h3>Captain(2x points) - {item.Captain}</h3>
        <h3>Vice Captain - {item.Vice_Captain}</h3>
        <h3>Predicted_Total_Points - {item.Predicted_Total_Points}</h3>
        <h3>Actual_Total_Points - {item.Actual_Total_Points}</h3>
        <h3>Total XI cost - {item.XI_Cost}</h3>
        <h3>Predicted_Bench_Points - {item.Predicted_Bench_Points}</h3>
        <h3>Bench_value - {item.Bench_value}</h3>
        <h3>Remaining_budget - {item.Remaining_budget}</h3>
      </div>
      <div className='middle'>
        <img src={myImage} alt="My Image" />
        <div className='fwd'>
          <div className='pointDiv'>
            <Tooltip className="ToolTip" title={item.Starting_XI.FWD[0].Team +
" , PredPoints = " + item.Starting_XI.FWD[0].Predicted_Points + " , Cost = " +
item.Starting_XI.FWD[0].Cost} arrow placement='top'>
              <div className={'point ' +
item.Starting_XI.FWD[0].TeamCSS}></div>
            </Tooltip>
            <p className='playerName'>{item.Starting_XI.FWD[0].Name}</p>
          </div>
          <div className='pointDiv'>
            <Tooltip className="ToolTip" title={item.Starting_XI.FWD[1].Team +
" , PredPoints = " + item.Starting_XI.FWD[1].Predicted_Points + " , Cost = " +
item.Starting_XI.FWD[1].Cost} arrow placement='top'>
              <div className={'point ' +
item.Starting_XI.FWD[1].TeamCSS}></div>
            </Tooltip>
            <p className='playerName'>{item.Starting_XI.FWD[1].Name}</p>
          </div>
        </div>
        <div className='mid'>
          <div className='upperMid'>

```

```

        <div className='pointDiv'>
            <Tooltip className="ToolTip" title={item.Starting_XI.MID[0].Team
+ " , PredPoints = " + item.Starting_XI.MID[0].Predicted_Points + " , Cost = " +
item.Starting_XI.MID[0].Cost} arrow placement='top'>
                <div className={'point ' +
item.Starting_XI.MID[0].TeamCSS}></div>
            </Tooltip>
            <p className='playerName'>{item.Starting_XI.MID[0].Name}</p>
        </div>
        <div className='pointDiv'>
            <Tooltip className="ToolTip" title={item.Starting_XI.MID[1].Team
+ " , PredPoints = " + item.Starting_XI.MID[1].Predicted_Points + " , Cost = " +
item.Starting_XI.MID[1].Cost} arrow placement='top'>
                <div className={'point ' +
item.Starting_XI.MID[1].TeamCSS}></div>
            </Tooltip>
            <p className='playerName'>{item.Starting_XI.MID[1].Name}</p>
        </div>
        <div className='pointDiv'>
            <Tooltip className="ToolTip" title={item.Starting_XI.MID[2].Team
+ " , PredPoints = " + item.Starting_XI.MID[2].Predicted_Points + " , Cost = " +
item.Starting_XI.MID[2].Cost} arrow placement='top'>
                <div className={'point ' +
item.Starting_XI.MID[2].TeamCSS}></div>
            </Tooltip>
            <p className='playerName'>{item.Starting_XI.MID[2].Name}</p>
        </div>
        <div className='lowerMid'>
            <div className='pointDiv'>
                <Tooltip className="ToolTip" title={item.Starting_XI.MID[3].Team
+ " , PredPoints = " + item.Starting_XI.MID[3].Predicted_Points + " , Cost = " +
item.Starting_XI.MID[3].Cost} arrow placement='top'>
                    <div className={'point ' +
item.Starting_XI.MID[3].TeamCSS}></div>
                </Tooltip>
                <p className='playerName'>{item.Starting_XI.MID[3].Name}</p>
            </div>
            <div className='pointDiv'>
                <Tooltip className="ToolTip" title={item.Starting_XI.MID[4].Team
+ " , PredPoints = " + item.Starting_XI.MID[4].Predicted_Points + " , Cost = " +
item.Starting_XI.MID[4].Cost} arrow placement='top'>
                    <div className={'point ' +
item.Starting_XI.MID[4].TeamCSS}></div>
                </Tooltip>
                <p className='playerName'>{item.Starting_XI.MID[4].Name}</p>
            </div>
        </div>
    </div>
    <div className='def'>

```

```

        <div className='pointDiv'>
            <Tooltip className="ToolTip" title={item.Starting_XI.DEF[0].Team +
" , PredPoints = " + item.Starting_XI.DEF[0].Predicted_Points + " , Cost = " +
item.Starting_XI.DEF[0].Cost} arrow placement='top'>
                <div className={'point ' +
item.Starting_XI.DEF[0].TeamCSS}></div>
                </Tooltip>
                <p className='playerName'>{item.Starting_XI.DEF[0].Name}</p>
            </div>
            <div className='pointDiv'>
                <Tooltip className="ToolTip" title={item.Starting_XI.DEF[1].Team +
" , PredPoints = " + item.Starting_XI.DEF[1].Predicted_Points + " , Cost = " +
item.Starting_XI.DEF[1].Cost} arrow placement='top'>
                    <div className={'point ' +
item.Starting_XI.DEF[1].TeamCSS}></div>
                    </Tooltip>
                    <p className='playerName'>{item.Starting_XI.DEF[1].Name}</p>
                </div>
            <div className='pointDiv'>
                <Tooltip className="ToolTip" title={item.Starting_XI.DEF[2].Team +
" , PredPoints = " + item.Starting_XI.DEF[2].Predicted_Points + " , Cost = " +
item.Starting_XI.DEF[2].Cost} arrow placement='top'>
                    <div className={'point ' +
item.Starting_XI.DEF[2].TeamCSS}></div>
                    </Tooltip>
                    <p className='playerName'>{item.Starting_XI.DEF[2].Name}</p>
                </div>
        </div>
        <div className='gk'>
            <div className='pointDiv'>
                <Tooltip className="ToolTip" title={item.Starting_XI.GK[0].Team + "
, PredPoints = " + item.Starting_XI.GK[0].Predicted_Points + " , Cost = " +
item.Starting_XI.GK[0].Cost} arrow placement='top'>
                    <div className={'point ' +
item.Starting_XI.GK[0].TeamCSS}></div>
                    </Tooltip>
                    <p className='playerName'>{item.Starting_XI.GK[0].Name}</p>
                </div>
            </div>
        </div>
        <div className='right'>
            <h2>Bench</h2>
            <div className='fwd'>
                <img src={bench} alt="My Image" />
                <div className='pointDiv'>
                    <Tooltip className="ToolTip" title={item.Bench[0].Team + " ,
PredPoints = " + item.Bench[0].Predicted_Points + " , Cost = " + item.Bench[0].Cost}
arrow placement='top'>
                        <div className={'point ' + item.Bench[0].TeamCSS}></div>
                        </Tooltip>
                        <p className='playerName'>{item.Bench[0].Name}</p>
                    </div>
                </div>
            </div>
        </div>
    </div>

```

```

</div>
<div className='pointDiv'>
  <Tooltip className="ToolTip" title={item.Bench[1].Team + " ,
PredPoints = " + item.Bench[1].Predicted_Points + " , Cost = " + item.Bench[1].Cost}
arrow placement='top'>
    <div className={'point ' + item.Bench[1].TeamCSS}></div>
  </Tooltip>
  <p className='playerName'>{item.Bench[1].Name}</p>
</div>
<div className='pointDiv'>
  <Tooltip className="ToolTip" title={item.Bench[2].Team + " ,
PredPoints = " + item.Bench[2].Predicted_Points + " , Cost = " + item.Bench[2].Cost}
arrow placement='top'>
    <div className={'point ' + item.Bench[2].TeamCSS}></div>
  </Tooltip>
  <p className='playerName'>{item.Bench[2].Name}</p>
</div>
<div className='pointDiv'>
  <Tooltip className="ToolTip" title={item.Bench[3].Team + " ,
PredPoints = " + item.Bench[3].Predicted_Points + " , Cost = " + item.Bench[3].Cost}
arrow placement='top'>
    <div className={'point ' + item.Bench[3].TeamCSS}></div>
  </Tooltip>
  <p className='playerName'>{item.Bench[3].Name}</p>
</div>
</div>
<div className='transfer'>
  <h3>Transferred for GameWeek</h3>
  {item.Transferred_Out ?
    <div className='transfers'>
      <div className='pointDiv'>
        <Tooltip className="ToolTip" title={item.Transferred_Out.Team
+ " , PredPoints = " + item.Transferred_Out.Predicted_Points + " , Cost = " +
item.Transferred_Out.Cost} arrow placement='top'>
          <div className={'point ' +
item.Transferred_Out.TeamCSS}></div>
        </Tooltip>
        <p className='playerName'>{item.Transferred_Out.Name}</p>
      </div>
      <div className='imgDiv'>
        <img src={swap} alt="My Image" />
      </div>
      <div className='pointDiv'>
        <Tooltip className="ToolTip" title={item.Transferred_In.Team +
" , PredPoints = " + item.Transferred_In.Predicted_Points + " , Cost = " +
item.Transferred_In.Cost} arrow placement='top'>
          <div className={'point ' +
item.Transferred_In.TeamCSS}></div>
        </Tooltip>
        <p className='playerName'>{item.Transferred_In.Name}</p>
      </div>
    </div>
  }

```

```

        </div>
        : "Not Available"}

    </div>
  </div>
</div>
));

return (
  <AliceCarousel
    autoPlay
    autoPlayInterval = {3000}
    infinite
    mouseTracking items={items} />
  );
}

export default Carousel;

```