CP317 Final Project - NBA Team Total Projection Betting Tool

Final Project Report

CP317

Wilfrid Laurier University

Javier Rodrigues 210631740

Shane Shamku 210844530

Inam Ul Haque 210631740

Vishwa Rajasegar 210218860

Maathulan Baskaran 210317900

Summary:

This project centers on the development of an NBA team total projection betting tool, leveraging advanced machine learning techniques to predict NBA team scores and provide betting recommendations. The tool aims to assist bettors in making informed decisions by generating precise projections based on historical data. The report begins with an overview of the project's motivation and objectives, emphasizing the challenges faced by sports bettors and the advantages of using statistical models to gain an edge over oddsmakers. It outlines the project's primary goals: developing an accurate predictive model, offering actionable betting recommendations, and creating a user-friendly web application.

The system architecture section details the top-level components of the project, including the frontend (built with React), the backend (using Node.js and Express), the database (MongoDB), and the predictive model (developed in Python). Each component's role and interactions are described, highlighting the seamless integration of the different parts of the system. The solution features section provides a detailed tabulation of the product features, mapping them to their respective components and functionalities.

Key features include intuitive user interfaces for selecting dates and viewing game projections, CRUD operations for managing game data, predictive modeling for generating team total projections, and performance tracking to continuously improve the model.

The report concludes with a reflection on the project's achievements and potential improvements. The NBA team total projection betting tool successfully integrates advanced predictive modeling with a robust web application, offering users valuable insights and recommendations for NBA betting. While the model shows promising results, continuous refinement and incorporation of additional data will further enhance its accuracy and utility.

Introduction:

Sports betting has long been a common and widespread practice among millions of people. However, on average, over time, people often lose money, and they lose big. Why is that? Are we not just making guesses about what events will happen? Why do we always seem to bet on the wrong side more times than not? And how do we, as oddsmakers, have a definitively better idea of outcomes in a high variance chance game? This is where statistical models and the oddsmakers edge come into play. The reason why 90% of sports bettors lose in the long run is because they are unable to project and predict sports outcomes to the same degree as the highly efficient and well trained models that generate the betting lines. Still, betting lines often do move multiple bps from open to close, even without external or game specific factors changing at all. How can this be if the oddsmakers models are so accurate? This is where oddsmakers margins, smart money, and true odds come into play. Generally, when putting out a spread, the oddsmakers can't put out a 100% fair and accurate line on both sides of a prop because if they did, they wouldn't be able to make money if both sides of the prop were bet relatively evenly. This is why oddsmakers have margins on one or both sides to shift odds and scrape away returns so that they make money no matter what. On top of all this, smart and experienced bettors with accurate projections can often find miscalculated/misprojected lines with their own deep analysis. When this happens, the smart money usually crowds onto one side of the betting line and thus, the oddsmakers shift the line towards the smart money side to try and encourage fair betting on both sides, usually applying a greater margin on the odds they provide.

This is all a long-winded way of saying that it's not impossible to outsmart or beat the oddsmakers. Although it may seem daunting, with the right objective approach, sharp projections can be calculated, which combined with personal research and due diligence, may give the bettor an edge over the books, especially for "juiced" or shifted lines. Therefore, our program aims to take a preliminary stab at this endeavour by projecting over/unders for NBA team total projections. Generally, each NBA game for the regular season and playoffs has a betting team total for both sides, our program aims to project an accurate team total for each team in each season between 2016 and 2022 using historical box score, advanced stat and historical odds data. Users will navigate through a web application where they can pick a date and game to view projections/betting recommendations for. As an example, consider Game 4 of the 2021 NBA finals where the Phoenix Suns played the Milwaukee Bucks. Consensus betting lines had the Bucks projected for 113 points in this game, our projection had them for 115.1 and they ended up scoring 119. In this case, betting the over of 112.5 with $100 would have netted you $89! Explanations for navigating through the program and

deriving these stats will be shown throughout the project, however, this is an illustrative example of the practicality of the program.

Problem Definition:

The core challenge our project seeks to address is the inherent complexity and difficulty of accurately predicting NBA team total points for betting purposes. Sports betting, particularly in high variance games like basketball, is fraught with uncertainty, making it difficult for the average bettor to consistently make profitable decisions. This difficulty arises from several key factors:

1. **High Variance in Game Outcomes:**
    - Basketball games are influenced by a multitude of dynamic factors, such as player performance, coaching strategies, injuries, and even psychological elements. This high variability makes it challenging to predict outcomes with precision.
2. **Efficient Oddsmaker Models:**
    - Oddsmakers use sophisticated statistical models and vast amounts of historical data to set betting lines. These models are designed to be highly efficient and accurate, making it difficult for casual bettors to find advantageous bets.
3. **Oddsmaker Margins and Smart Money:**
    - Oddsmakers incorporate margins into their lines to ensure profitability regardless of the bet outcomes. Additionally, experienced bettors ("smart money") can influence line movements by identifying and betting on mispriced lines. This can further complicate the betting landscape for the average bettor.
4. **Data Overload:**
    - The sheer volume of available data on player statistics, game outcomes, and historical odds can be overwhelming. Bettors often struggle to process and analyze this data effectively to make informed decisions.

Given these challenges, our project aims to leverage advanced statistical modeling and machine learning techniques to provide more accurate NBA team total projections. By analyzing historical box scores, advanced stats, and historical odds data, we seek to develop a model that can offer actionable insights and betting recommendations.

Objectives of the Project

1. **Develop an Accurate Prediction Model:**

    - Create a predictive model that can forecast NBA team total points with high accuracy. This model will be trained on historical game data, including box scores and advanced statistics, to capture the nuances of team performance.
2. **Provide Betting Recommendations:**
    - Offer clear and actionable betting recommendations based on the model's projections. These recommendations will include whether to bet over or under the set team totals, along with a rating to indicate the confidence level of each bet.
3. **User-Friendly Interface:**
    - Design and implement a web application that allows users to easily navigate through different games and dates. Users should be able to view projected team totals, compare them with the oddsmaker lines, and see the model's betting recommendations.
4. **Continuous Improvement:**
    - Continuously refine the model by incorporating new data and feedback. This will involve regular updates to the historical data used for training, as well as adjustments to the model parameters to enhance its predictive accuracy.

Scope and Limitations

1. **Scope:**
    - The project will focus on NBA regular season and playoff games between the 2016 and 2022 seasons. The model will be trained on historical data from this period and will provide projections for each game within this timeframe.
2. **Limitations:**
    - The accuracy of the model is inherently limited by the quality and completeness of the historical data. Unexpected events, such as sudden player injuries or changes in team dynamics, can also impact the model's predictions.
    - The project will not cover betting strategies beyond team total projections. While the model aims to provide an edge in betting, it does not guarantee profits and should be used as one of many tools in a bettor's decision-making process.

System Requirements

Functional Requirements:

User Interface and Navigation:

The application must provide an intuitive and user-friendly interface that allows users to navigate through different NBA seasons, months, and days. Users should be able to select a specific date and view a list of games scheduled for that day. Upon selecting a game, users should be directed to a detailed game view page where they can see the predicted team totals, betting lines, and odds for both teams.

Data Management and Integration:

The application must integrate with a MongoDB database to store and retrieve game data, predictions, and odds. The database should include collections for storing historical game logs, prediction results, and odds information. This information must be scraped, collected and formatted from online databases and exported to csv form for importing to the Mongo Web App. The system must also support CRUD (Create, Read, Update, Delete) operations for managing game data and predictions within the database.

Predictive Modeling:

The core functionality involves generating accurate team total projections using a machine learning model. The model must be trained on historical NBA data, including box scores, advanced stats, and historical odds.The system should provide endpoints for fetching model predictions based on specific parameters such as team, opponent, date, and season.

Betting Recommendations:

The application should calculate and display betting recommendations based on the model's projections and the odds provided by oddsmakers. Recommendations should include whether to bet over or under the team total, along with a confidence rating. The system should highlight discrepancies between the model’s predictions and the oddsmaker lines, suggesting potential value bets.

Performance Tracking:

The system must track the performance of its predictions by comparing them to actual game results. This feature should allow for continuous improvement of the predictive model based on historical accuracy.

Non-functional Requirements

1. Performance, Reliability & Robustness:

The system should deliver fast response times for user interactions, including data retrieval, predictions, and navigation. The goal is to maintain response times under two seconds for all user actions. The program should have a MTBF (mean time between failures) of atleast 100 user actions per failure and said failures should reflect a MTTR (mean time to repair) of under 1 human hour per failure. Should meet computer system constraints of under 100GB for the entire program and be able to run on Windows, Linux and OS as long as Python, Jupyter Labs, Statistical Libraries and MERN is working on those systems.

Utility & Correctness:

The system must be reliable, ensuring minimal downtime and consistent performance. It should be available to users 24/7, with planned maintenance causing minimal disruption. The user interface should be intuitive and easy to use, even for those with minimal technical knowledge. Design principles such as simplicity, consistency, and accessibility should be followed. Code must follow output specifications with structured readable and consistent output that is easily decipherable.

Maintainability:

The codebase should be well-documented and organized to facilitate maintenance and updates. This includes using modular code structures, clear naming conventions, and comprehensive inline comments.

Constraints and Assumptions + Solution Limitations and Assumptions

Data Quality and Availability:

It is assumed that the historical NBA data, including box scores and advanced statistics, is accurate and complete. Any discrepancies or missing data could impact the model's performance. Missing data from basketball reference or the historical odds online database is filled with NaN values and ultimately removed from final datasets. Games without historical odds/stats from which to derive predictions are thus dealt by with error handling and are unavailable on the app.

Technical Stack:

The application is built using a MERN stack (MongoDB, Express, React, Node.js). Constraints related to these technologies, such as compatibility issues and performance limitations, must be considered.

Model Limitations:

The predictive model is inherently limited by the historical data and the chosen machine learning techniques. It does not account for all variables affecting game outcomes, such as player injuries, tracking data or individual player box scores and advanced stats. Only game high player stats for each statistical category are used in the predictions, not all player stats. For example in the 2016 NBA finals game 7 when LeBron James scored 27 points and Kyrie Irving scored 26 points. Only LeBron's game high 27 points is included in the dataset for which our predictions are then calculated off of. The model also requires 2 seasons worth of data before it can generate reliable predictions, thus predictions for the 2015-16 and 2017-18 NBA seasons are unavailable.

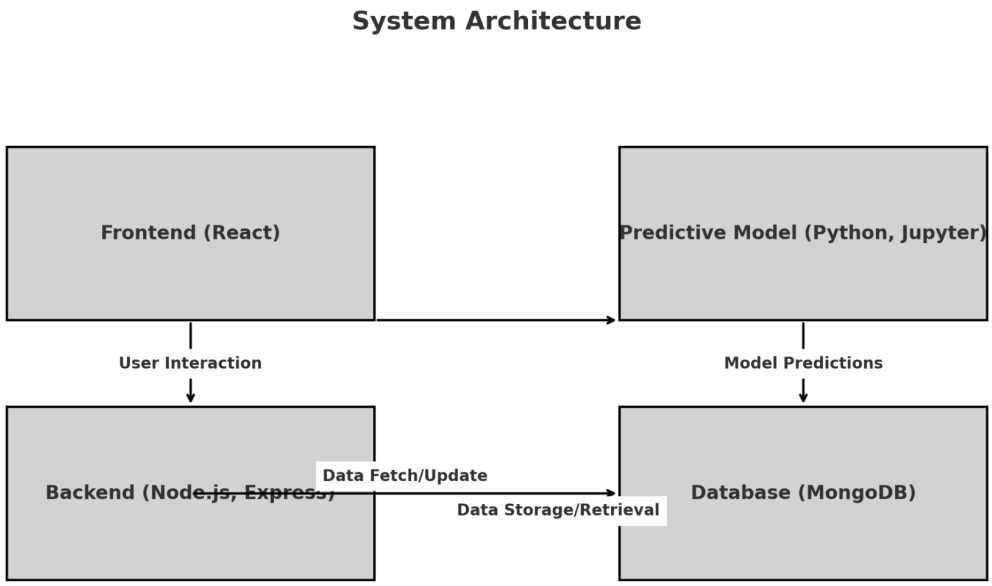
User Base:

The application is designed with the assumption that the primary users are individuals interested in NBA betting. User feedback and usage patterns may vary, affecting the system's perceived value and usability. The program is not designed to be used solely and exclusively when placing bets, it is not meant to be a comprehensive recommendation, rather it is supposed to be a basis or guide for which the user must then conduct their own research and due diligence before placing any bets related to the projected props.

Project Timeline:

Development is constrained by the project timeline, requiring efficient use of resources and adherence to deadlines to ensure timely delivery of all features and functionalities. In this case, the requirements, analysis and design phases were conducted over May and June and the implementation phase, testing and post-maintenance phases have taken place over July. Relatively speaking this is not a lot of time to code a comprehensive project, so there are limitations in terms of code scope, features etc.

System Architecture



Proposed Top-Level Components

Frontend (Client):

- Language/Specs: React
- Role: The frontend provides an intuitive and interactive user interface, allowing users to select a date and view a list of NBA games for that day. Upon selecting a game, users can see detailed predictions and betting recommendations.
- Key Components:

        - `App.js`: Main application component that sets up routing and provides the overall layout.
        - `Dropdown.js`: Component for dropdown menus allowing users to select the season, month, and day.
        - `GamesList.js`: Displays a list of games based on the selected date.
        - `GameDetails.js`: Shows detailed predictions and betting recommendations for a selected game.

Backend (Server):

- Language/Specs: Node.js, Express
- Role: The backend handles core business logic, data retrieval, predictive modeling, and serves API endpoints to the frontend.
- Key Functionalities:
    - API Endpoints: Provides endpoints for fetching game data, predictions, and odds.
    - Database Integration: Connects to MongoDB to store and retrieve game logs, predictions, and odds data.
    - Predictive Model Integration: Interfaces with the predictive model to generate and return team total projections.
- File: `app.js`: Main server file that sets up the Express server, connects to MongoDB, defines the schema, and handles API routes.

Database:

- Language/Specs: MongoDB
- Role: The database stores historical game logs, prediction results, and odds information. It supports CRUD operations for managing game data and predictions.
- Schema: The `gameSchema` defined in `app.js` maps the structure of game data in MongoDB, including fields for team names, dates, predictions, and actual scores.

Predictive Model:

- Language/Specs: Python, Jupyter Notebook
- Role: The predictive model leverages historical NBA data to generate team total projections. It uses machine learning techniques to analyze past performance and predict future outcomes.
- Key Files:
  - `predict.ipynb`: Jupyter notebook containing the machine learning model and training logic.
  - `scrapeNBAstats.py`: Script for scraping historical NBA stats from relevant sources and preparing the data for model training.

Interfaces between Components

Client-Server Interaction:

The frontend communicates with the backend via RESTful API calls. When a user selects a date and game, the client sends a request to the server to fetch the relevant game data and predictions. The server processes the request, retrieves the necessary data from MongoDB, runs the predictive model, and returns the results to the client.

Server-Database Interaction:

The server uses Mongoose to interact with MongoDB. It performs CRUD operations to store and retrieve game logs, predictions, and odds data. The `gameSchema` defined in `app.js` maps the structure of game data in MongoDB.

Server-Model Interaction:

The server integrates with the predictive model by calling Python scripts or REST endpoints exposed by the model. The model processes historical data and generates team total projections, which the server then uses to provide betting recommendations.

Solution Features

Tabulated Product Features

Feature	Description	Components Used
User Interface	Intuitive and user-friendly interface for selecting dates and viewing game predictions	`App.js`, `Dropdown.js`, `GamesList.js`, `GameDetails.js`
Date and Game Selection	Dropdown menus to select NBA season, month, and day	`Dropdown.js` - `handleSeasonChange()`, `handleMonthChange()`, `handleDayChange()`
Game List Display	Display list of NBA games for the selected date	`GamesList.js` - `fetchGames()`, `renderGames()`
Game Details View	Show detailed predictions and betting recommendations for a selected game	`GameDetails.js` - `fetchGameDetails()`, `renderGameDetails()`
API Endpoints	Provide endpoints for fetching game data, predictions, and odds	`app.js` - `get('/api/games')`, `get('/api/game/:id')`

Database Integration	Store and retrieve game logs, predictions, and odds	`app.js` - `connectToDatabase()`, `gameSchema`

CRUD Operations	Support for Create, Read, Update, Delete operations for game data and predictions	`app.js` - `createGame()`, `readGame()`, `updateGame()`, `deleteGame()`

Predictive Modeling	Generate team total projections using machine learning	`predict.ipynb` - `trainModel()`, `predict()`, `evaluateModel()`

Data Scraping	Scrape historical NBA stats and odds from relevant sources	`scrapeNBAstats.py` - `scrapeStats()`, `cleanData()`, `saveData()`

Performance Tracking	Track performance of predictions by comparing to actual game results	`app.js` - `comparePredictions()`, `logResults()`

Bet Rating Calculation	Calculate and display bet ratings based on model predictions and betting lines	`GameDetails.js` - `calculateBetRating()`, `renderBetRating()`

Security and Authentication	Implement security measures to protect user data and ensure secure access	`app.js` - `authenticateUser()`, `authorizeUser()`, `encryptData()`

Final Functionalities of Top-Level & Sub Components

Frontend (React):

- User Interface: Provides a clean and intuitive interface for users to interact with the application. Components like Dropdown.js, GamesList.js, and GameDetails.js ensure seamless navigation and display of data.
  - Functions:
    - App.js: Main application component that sets up routing and provides the overall layout.
    - Dropdown.js:
      - handleSeasonChange(): Handles season dropdown changes.
      - handleMonthChange(): Handles month dropdown changes.
      - handleDayChange(): Handles day dropdown changes.
    - GamesList.js:
      - fetchGames(): Fetches list of games for the selected date.
      - renderGames(): Renders the list of games.
    - GameDetails.js:
      - fetchGameDetails(): Fetches detailed information for a selected game.
      - renderGameDetails(): Renders game details including predictions and betting lines.
      - calculateBetRating(): Calculates the bet rating based on predictions and betting lines.
      - renderBetRating(): Renders the bet rating.

Backend (Node.js, Express):

- API Endpoints: Provides RESTful API endpoints to fetch game data, predictions, and odds. These endpoints serve as the communication bridge between the frontend and the database/predictive model.
  - Functions:
    - app.js:
      - get('/api/games'): Endpoint to get a list of games.
      - get('/api/game/:id'): Endpoint to get details of a specific game.

- Database Integration: Connects to MongoDB to store and retrieve game logs, predictions, and odds data. Implements Mongoose for schema definition and CRUD operations.
    - Functions:
        - connectToDatabase(): Establishes connection to MongoDB.
        - gameSchema: Defines the schema for game data.
- CRUD Operations: Support for Create, Read, Update, Delete operations for game data and predictions.
    - Functions:
        - createGame(): Creates a new game entry in the database.
        - readGame(): Reads a game entry from the database.
        - updateGame(): Updates an existing game entry in the database.
        - deleteGame(): Deletes a game entry from the database.
- Performance Tracking: Tracks the performance of predictions by comparing them to actual game results, allowing continuous improvement of the predictive model.
    - Functions:
        - comparePredictions(): Compares model predictions with actual results.
        - logResults(): Logs the performance results for analysis.

Predictive Model (Python, Jupyter):

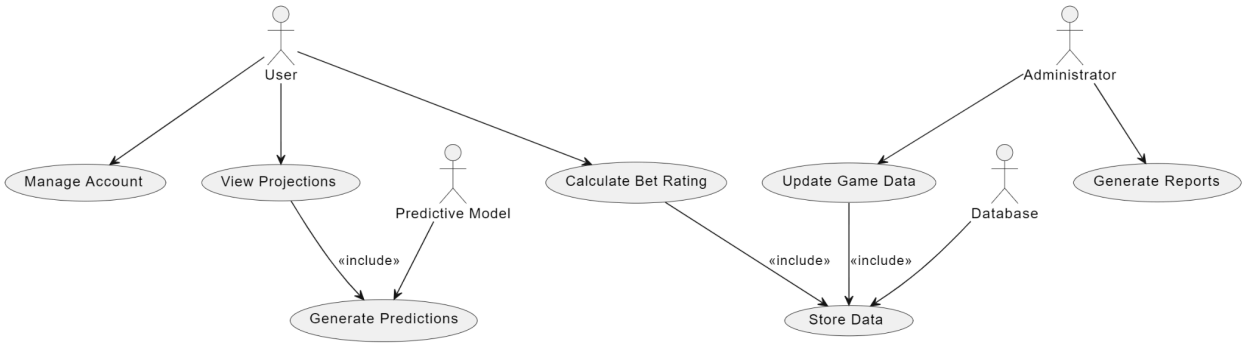
- Predictive Modeling: Leverages historical NBA data to generate team total projections using machine learning techniques. This model is continuously improved by incorporating new data and refining its algorithms.
    - Functions:
        - predict.ipynb:
            - trainModel(): Trains the predictive model using historical data.
            - predict(): Generates predictions for upcoming games.
            - evaluateModel(): Evaluates the performance of the model.
- Data Scraping: Scrapes historical NBA stats and odds from relevant sources, preparing the data for model training. This ensures that the model has access to accurate and comprehensive data for making predictions.
    - Functions:
        - scrapeNBAstats.py:
            - scrapeStats(): Scrapes historical NBA statistics.

- cleanData(): Cleans and processes the scraped data.
- saveData(): Saves the cleaned data for model training.

Component Diagrams

- Internal Structures of Two Major Components

Use Case Diagram



Actors

1. User: The primary actor who interacts with the system to view game projections, betting lines, and betting recommendations.
2. Administrator: An actor responsible for managing the system, updating game data, and monitoring the predictive model's performance.
3. Predictive Model: An external system that generates team total projections based on historical data.
4. Database: An external system that stores game data, predictions, and betting lines.

Use Cases

1. View Projections:
    - Actor: User
    - Description: The user selects a date and game to view detailed projections for the teams involved, including betting lines, odds, and bet recommendations.
    - Preconditions: The user has accessed the application and navigated to the game selection page.
    - Postconditions: The user views the projections and recommendations for the selected game.

2. Calculate Bet Rating:
    - Actor: User
    - Description: The system calculates the bet rating based on the model's prediction and the betting lines provided by the user.
    - Preconditions: The user has selected a game and the system has fetched the necessary data.
    - Postconditions: The system displays the calculated bet rating and the corresponding recommendation.
3. Manage Account:
    - Actor: User
    - Description: The user manages their account details, including updating personal information and viewing betting history.
    - Preconditions: The user is logged into their account.
    - Postconditions: The account details are updated, and the user can view their betting history.
4. Generate Reports:
    - Actor: Administrator
    - Description: The administrator generates reports on the system's performance, including the accuracy of predictions and user betting trends.
    - Preconditions: The administrator is logged into the system.
    - Postconditions: The system generates and displays the requested reports.
5. Update Game Data:
    - Actor: Administrator
    - Description: The administrator updates the game data, including historical stats and betting lines.
    - Preconditions: The administrator is logged into the system.
    - Postconditions: The game data is updated, and the predictive model is retrained if necessary.
6. Generate Predictions:
    - Actor: Predictive Model
    - Description: The predictive model generates team total projections based on historical data.
    - Preconditions: The system has provided the necessary historical data to the predictive model.
    - Postconditions: The predictive model returns the projections to the system.
7. Store Data:
    - Actor: Database

- Description: The database stores game data, predictions, and betting lines.
- Preconditions: The system has provided the data to be stored.
- Postconditions: The data is stored in the database and is available for retrieval.

Details of Two Use Cases

1. *View Projections*:
    - Actor: User
    - Description: The user selects a date and game to view detailed projections for the teams involved, including betting lines, odds, and bet recommendations.
    - Steps:
        1. The user navigates to the game selection page.
        2. The user selects a date using the dropdown menus.
        3. The user selects a game from the list of games displayed for the chosen date.
        4. The system fetches the game details, including team projections, betting lines, and odds.
        5. The system calculates the bet rating based on the projections and odds.
        6. The system displays the game details, projections, and bet recommendations to the user.
    - Preconditions: The user has accessed the application and navigated to the game selection page.
    - Postconditions: The user views the projections and recommendations for the selected game.

2. *Calculate Bet Rating*:
    - Actor: User
    - Description: The system calculates the bet rating based on the model's prediction and the betting lines provided by the user.
    - Steps:
        1. The user selects a game and views the projections and recommendations.
        2. The system retrieves the betting lines and model predictions for the selected game.

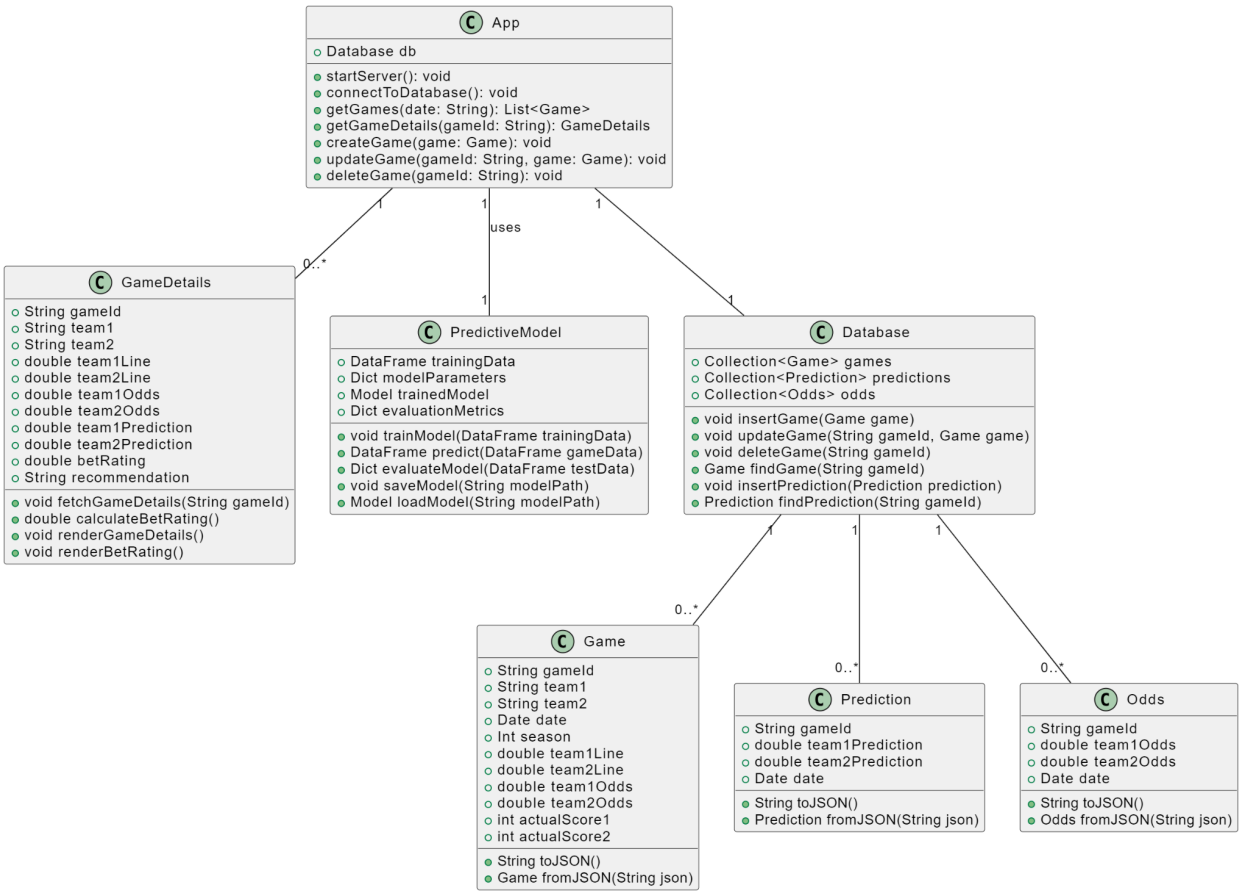
3. The system calculates the bet rating based on the difference between the model prediction and the betting lines.
4. The system takes into account the odds for higher and lower bets to refine the bet rating.
5. The system assigns a verbal recommendation based on the bet rating.
6. The system displays the calculated bet rating and the recommendation to the user.

- Preconditions: The user has selected a game and the system has fetched the necessary data.
- Postconditions: The system displays the calculated bet rating and the corresponding recommendation.

Mapping Use Cases to Product Features

Use Case	Product Feature
View Projections	Date and Game Selection, Game List Display, Game Details View, Predictive Modeling, Bet Rating Calculation
Calculate Bet Rating	Bet Rating Calculation, Predictive Modeling
Manage Account	User Authentication, Manage Account
Generate Reports	Performance Tracking
Update Game Data	Database Integration, Data Scraping, Predictive Modeling
Generate Predictions	Predictive Modeling
Store Data	Database Integration

UML Class Diagram



Explanation of Relationships and Responsibilities

App (Backend):

- Attributes: Contains a reference to the Database instance.
- Operations: Handles server startup, database connections, and CRUD operations for games.
- Relationships:
  - uses the PredictiveModel to generate predictions.
  - Has a one-to-many relationship with GameDetails, as multiple game details can be fetched through the app.
  - Has a one-to-one relationship with Database, as the app interacts with a single database instance.

GameDetails (Frontend):

- Attributes: Stores information related to the game and its predictions.

- Operations: Fetches game details, calculates bet ratings, and renders information.
- Relationships: Uses the PredictiveModel indirectly via the app to fetch predictions.

PredictiveModel (Model):

- Attributes: Contains data and parameters needed for training and prediction.
- Operations: Trains the model, generates predictions, evaluates performance, saves, and loads models.
- Relationships: Interacts with the app to provide predictions.

Database (MongoDB):

- Attributes: Contains collections for Game, Prediction, and Odds.
- Operations: Performs CRUD operations for games, predictions, and odds.
- Relationships:
  - Has a one-to-many relationship with Game, Prediction, and Odds.

Game:

- Attributes: Stores game-specific information.
- Operations: Serializes and deserializes game data to and from JSON.

Prediction:

- Attributes: Stores prediction-specific information.
- Operations: Serializes and deserializes prediction data to and from JSON.

Odds:

- Attributes: Stores odds-specific information.
- Operations: Serializes and deserializes odds data to and from JSON.

Decision Rationale:

Cohesion:

Each class is highly cohesive, focusing on a single responsibility. For example, GameDetails handles game-specific information, while PredictiveModel focuses on prediction tasks. This ensures that each class has a clear and focused purpose, which simplifies maintenance and understanding.

Coupling:

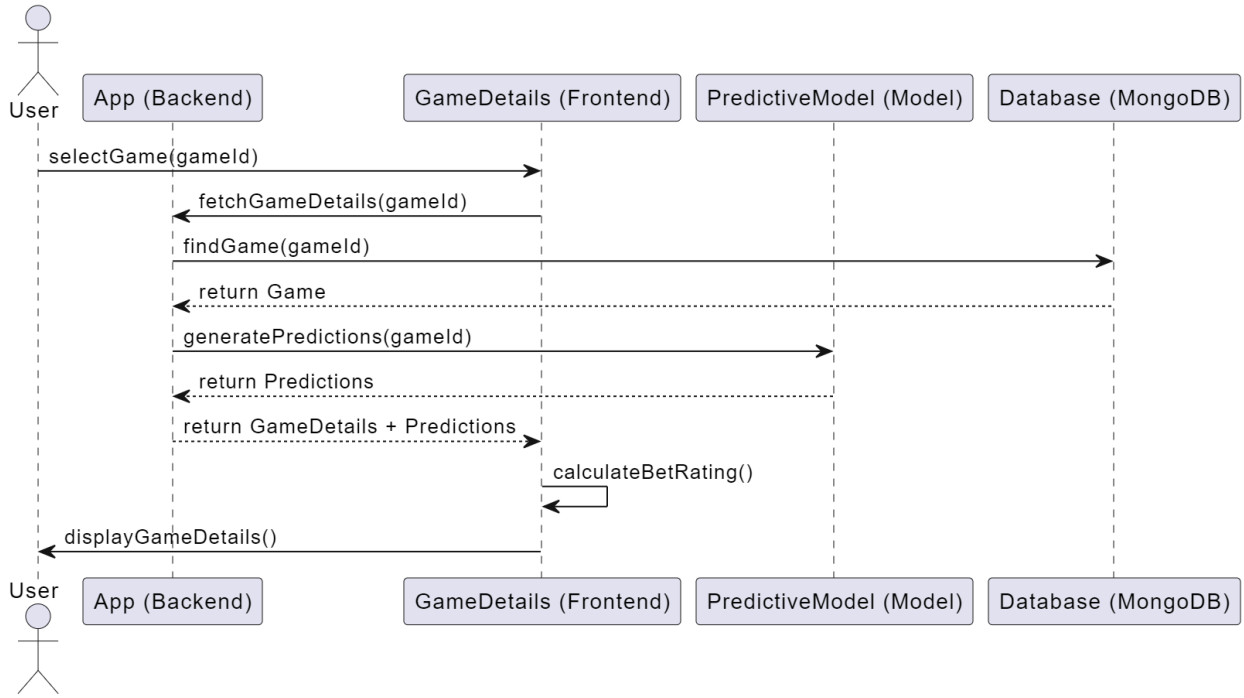
Coupling is minimized by defining clear interfaces for interactions between components. For instance, the app interacts with the database and predictive model through well-defined methods, reducing direct dependencies. This modularity allows for easier updates and changes without affecting unrelated parts of the system.

Code Reuse:

The design promotes code reuse by encapsulating common functionalities within specific classes. For example, Database handles all database operations, which can be reused across different parts of the application without duplicating code.

Sequence Diagrams

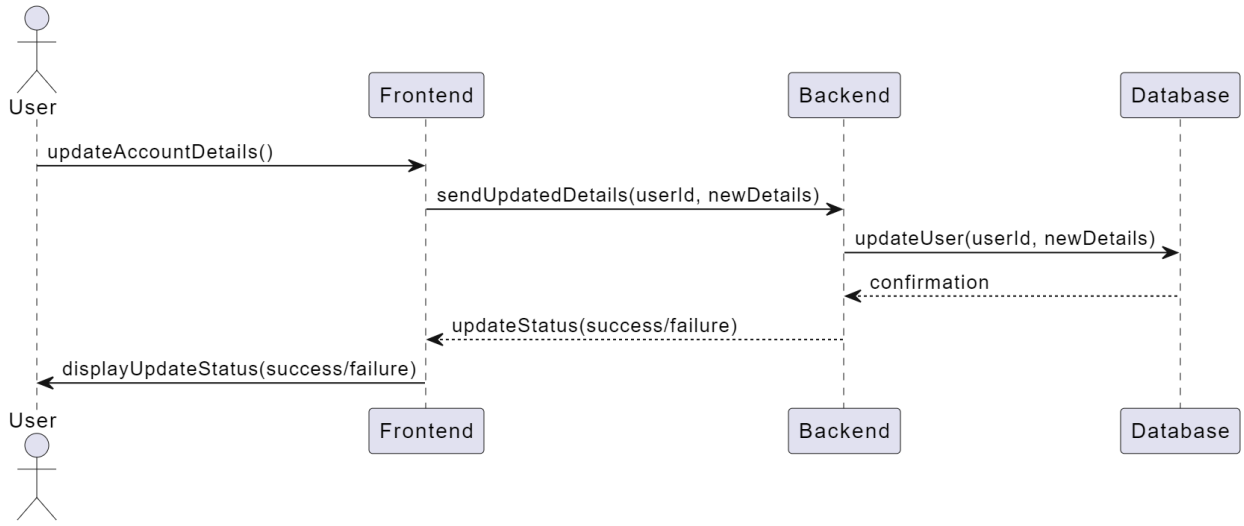
Sequence Diagram 1: View Projections



1. User selects a game:
    - Message: `User -> GD: selectGame(gameId)`
    - Description: The user interacts with the frontend to select a game by its gameId.
2. Frontend requests game details from backend:
    - Message: `GD -> App: fetchGameDetails(gameId)`

    - Description: The frontend sends a request to the backend to fetch the game details for the selected game.
3. Backend queries the database:
    - Message: `App -> DB: findGame(gameId)`
    - Description: The backend queries the MongoDB database to retrieve the game details using the gameId.
4. Database returns game details:
    - Message: `DB --> App: return Game`
    - Description: The database returns the game details to the backend.
5. Backend requests predictions from predictive model:
    - Message: `App -> PM: generatePredictions(gameId)`
    - Description: The backend requests the predictive model to generate predictions for the selected game.
6. Predictive model returns predictions:
    - Message: `PM --> App: return Predictions`
    - Description: The predictive model returns the generated predictions to the backend.
7. Backend returns game details and predictions to frontend:
    - Message: `App --> GD: return GameDetails + Predictions`
    - Description: The backend sends the game details and predictions back to the frontend.
8. Frontend calculates bet rating:
    - Message: `GD -> GD: calculateBetRating()`
    - Description: The frontend calculates the bet rating based on the received game details and predictions.
9. Frontend displays game details to user:
    - Message: `GD -> User: displayGameDetails()`
    - Description: The frontend displays the detailed game information, including predictions and bet ratings, to the user.

Sequence Diagram 2: Manage Account



1. User updates account details:
    - Message: `User -> FE: updateAccountDetails()`
    - Description: The user interacts with the frontend to update their account details.
2. Frontend sends updated details to backend:
    - Message: `FE -> BE: sendUpdatedDetails(userId, newDetails)`
    - Description: The frontend sends the updated user details to the backend for processing.
3. Backend updates user details in database:
    - Message: `BE -> DB: updateUser(userId, newDetails)`
    - Description: The backend sends a request to the database to update the user's details with the new information.
4. Database confirms the update:
    - Message: `DB --> BE: confirmation`
    - Description: The database confirms that the user details have been successfully updated.
5. Backend sends update status to frontend:
    - Message: `BE --> FE: updateStatus(success/failure)`
    - Description: The backend sends the status of the update (success or failure) back to the frontend.
6. Frontend displays update status to user:
    - Message: `FE -> User: displayUpdateStatus(success/failure)`

- Description: The frontend displays the update status to the user, indicating whether the update was successful or not.

Design Rationale

Cohesion:

Each component in the sequence diagram has a well-defined role, ensuring high cohesion. For example, `GameDetails` focuses on displaying game details and calculating bet ratings, while `PredictiveModel` handles predictions, and the `User` component handles account updates. This separation of concerns ensures that each class has a single responsibility, making the system easier to maintain and extend.

Coupling:

Coupling is minimized by defining clear interfaces for interactions between components. For instance, the frontend interacts with the backend through specific API calls, and the backend interacts with the database and predictive model through well-defined methods. This reduces dependencies between components, making it easier to update or replace individual parts without affecting the rest of the system.

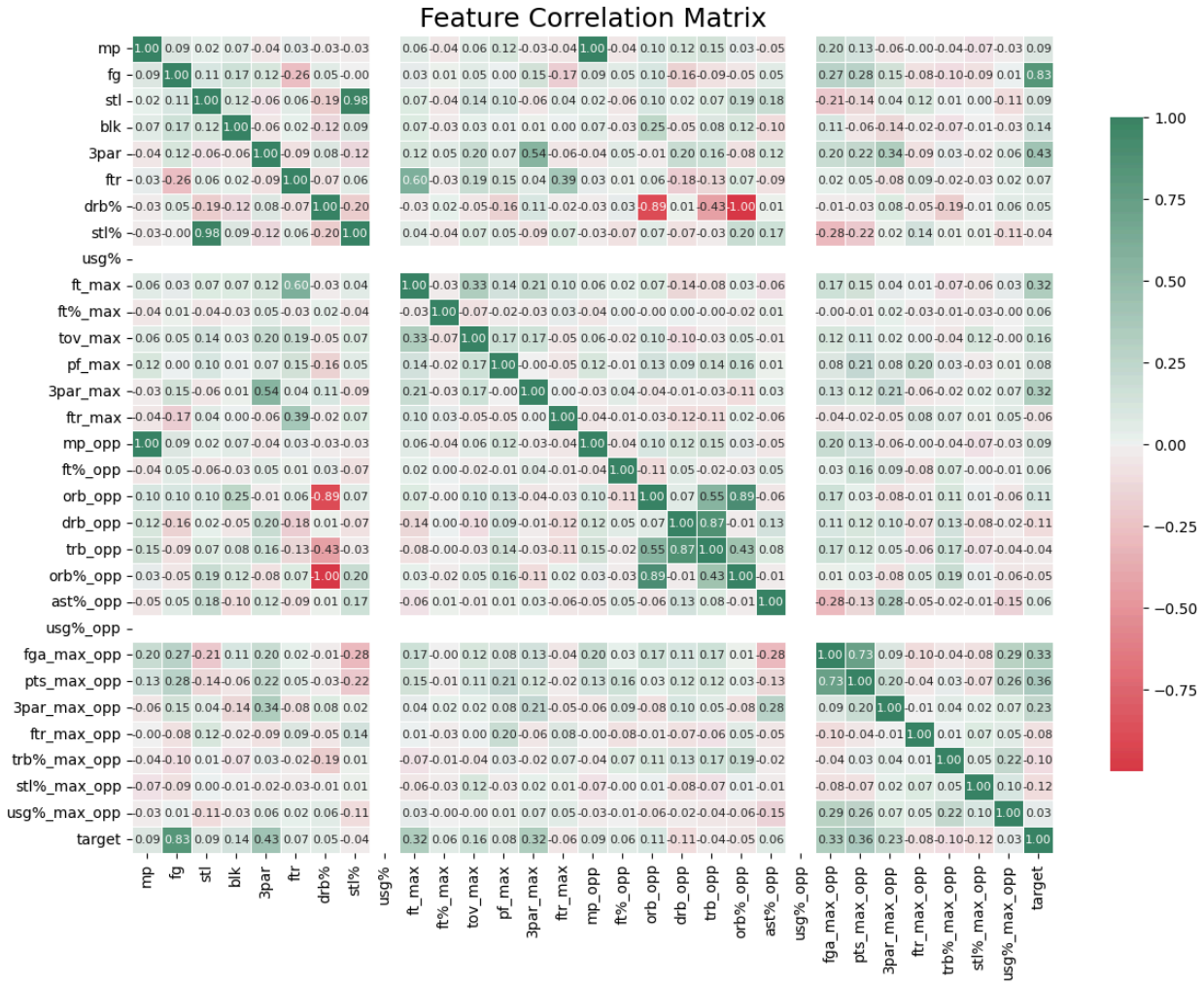
Efficiency Considerations:

The sequence diagrams ensure efficient data flow and processing. Data is fetched and processed only when needed, and intermediate results are passed directly to the next step in the sequence. This minimizes unnecessary data transfers and ensures that the system can respond quickly to user actions.

Appendices & Program Screenshots

Predict.ipynb:

Feature Selection Correlation Matrix:



Predictions Dataframe (Post Ridge Regression):

```
adjusted_predictions
✓ 0.0s
```

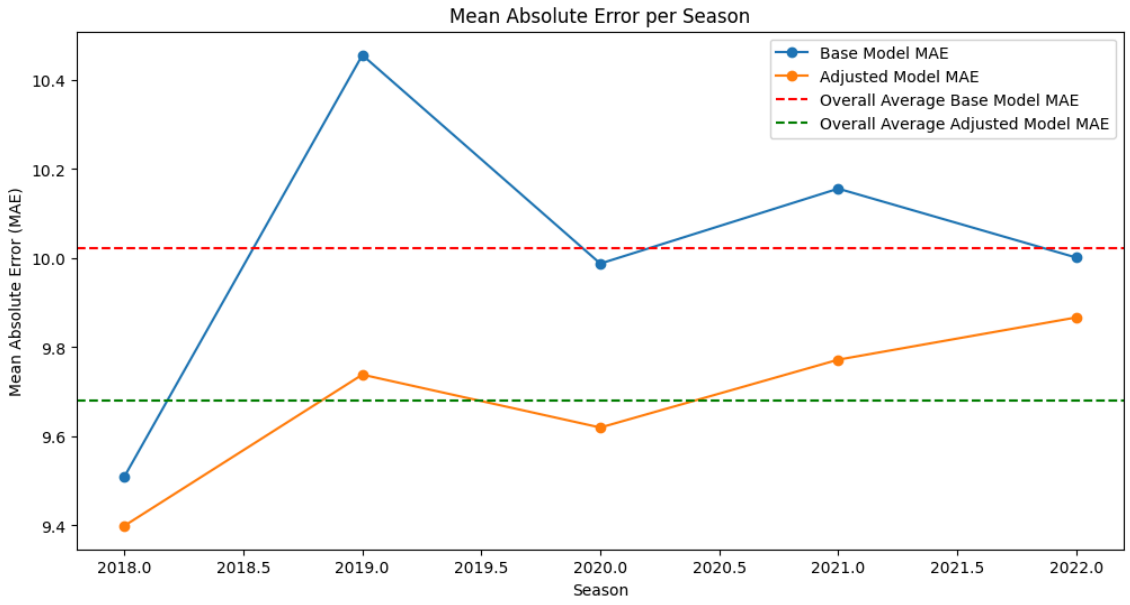
	team	season	team_opp	actual	date	prediction	Team_FullName	Team_FullName_Opp	prediction adj
5010	PHI	2018	HOU	112	2017-10-25	106.134149	Philadelphia 76ers	Houston Rockets	106.449661
5011	IND	2018	OKC	97	2017-10-25	103.622134	Indiana Pacers	Oklahoma City Thunder	104.522413
5012	HOU	2018	PHI	109	2017-10-25	111.044222	Houston Rockets	Philadelphia 76ers	107.782653
5013	BRK	2018	CLE	86	2017-10-25	108.652297	Brooklyn Nets	Cleveland Cavaliers	107.494931
5014	PHO	2018	UTA	107	2017-10-25	102.216841	Phoenix Suns	Utah Jazz	101.538037
...	...	...	...	...	...	...	...	...	...
16927	BOS	2022	GSW	94	2022-06-10	110.221772	Boston Celtics	Golden State Warriors	107.333140
16928	GSW	2022	BOS	103	2022-06-13	108.799706	Golden State Warriors	Boston Celtics	106.550560
16929	BOS	2022	GSW	90	2022-06-13	107.808226	Boston Celtics	Golden State Warriors	107.899757
16930	BOS	2022	GSW	100	2022-06-16	107.923640	Boston Celtics	Golden State Warriors	107.875244
16931	GSW	2022	BOS	100	2022-06-16	113.279320	Golden State Warriors	Boston Celtics	105.395918

11922 rows × 9 columns

Shifting & Scaling Calibration Testing:

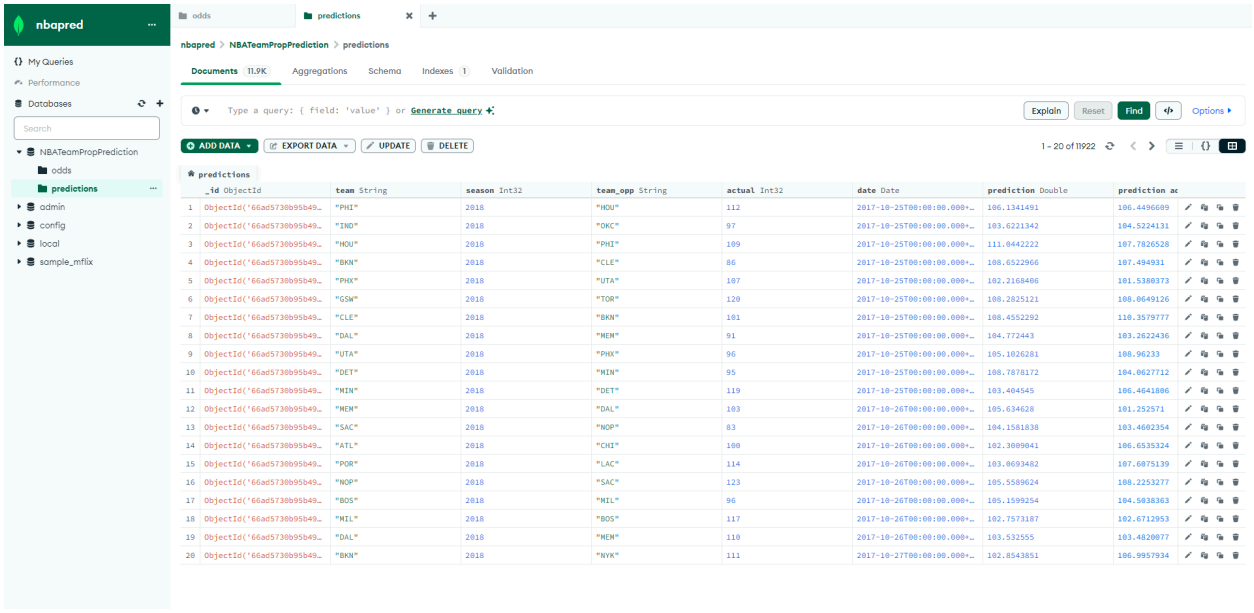
```
Team: Golden State Warriors, Opponent: Dallas Mavericks, season: 2022, Team ORtg: 112.5, Opp DRtg: 109.4, Pace: 96.9, prediction: 109.67443939258993, expected PPG: 107.51055000000001, inflator: 0.9757343973950889
Team: Dallas Mavericks, Opponent: Golden State Warriors, season: 2022, Team ORtg: 112.8, Opp DRtg: 106.9, Pace: 96.9, prediction: 112.30250079425313, expected PPG: 106.44465, inflator: 0.9353247620972827
Team: Miami Heat, Opponent: Boston Celtics, season: 2022, Team ORtg: 113.7, Opp DRtg: 106.9, Pace: 96.25, prediction: 110.43259269336099, expected PPG: 106.16375000000002, inflator: 0.9524699488284859
Team: Boston Celtics, Opponent: Miami Heat, season: 2022, Team ORtg: 114.4, Opp DRtg: 109.1, Pace: 96.25, prediction: 111.37474118355325, expected PPG: 107.559375, inflator: 0.9574370884764656
Team: Boston Celtics, Opponent: Miami Heat, season: 2022, Team ORtg: 114.4, Opp DRtg: 109.1, Pace: 96.25, prediction: 107.14453375539324, expected PPG: 107.559375, inflator: 1.004663090047296
Team: Miami Heat, Opponent: Boston Celtics, season: 2022, Team ORtg: 113.7, Opp DRtg: 106.9, Pace: 96.25, prediction: 106.79217373881316, expected PPG: 106.16375000000002, inflator: 0.9929369344763822
Team: Boston Celtics, Opponent: Golden State Warriors, season: 2022, Team ORtg: 114.4, Opp DRtg: 106.9, Pace: 97.5, prediction: 114.81442176335182, expected PPG: 107.88375, inflator: 0.9240241526232951
Team: Golden State Warriors, Opponent: Boston Celtics, season: 2022, Team ORtg: 112.5, Opp DRtg: 106.9, Pace: 97.5, prediction: 112.81606101376649, expected PPG: 106.9575, inflator: 0.9353172015246926
Team: Golden State Warriors, Opponent: Boston Celtics, season: 2022, Team ORtg: 112.5, Opp DRtg: 106.9, Pace: 97.5, prediction: 110.71496067067676, expected PPG: 106.9575, inflator: 0.9580733901400573
Team: Boston Celtics, Opponent: Golden State Warriors, season: 2022, Team ORtg: 114.4, Opp DRtg: 106.9, Pace: 97.5, prediction: 107.85193467418343, expected PPG: 107.88375, inflator: 1.0003576576992095
Team: Golden State Warriors, Opponent: Boston Celtics, season: 2022, Team ORtg: 112.5, Opp DRtg: 106.9, Pace: 97.5, prediction: 111.38343739817444, expected PPG: 106.9575, inflator: 0.9507561629794772
Team: Boston Celtics, Opponent: Golden State Warriors, season: 2022, Team ORtg: 114.4, Opp DRtg: 106.9, Pace: 97.5, prediction: 112.60174080866884, expected PPG: 107.88375, inflator: 0.947580221092107
Team: Golden State Warriors, Opponent: Boston Celtics, season: 2022, Team ORtg: 112.5, Opp DRtg: 106.9, Pace: 97.5, prediction: 113.33610604836491, expected PPG: 106.9575, inflator: 0.9298086357391223
Team: Boston Celtics, Opponent: Golden State Warriors, season: 2022, Team ORtg: 114.4, Opp DRtg: 106.9, Pace: 97.5, prediction: 110.22177172167362, expected PPG: 107.88375, inflator: 0.9737925522965285
Team: Golden State Warriors, Opponent: Boston Celtics, season: 2022, Team ORtg: 112.5, Opp DRtg: 106.9, Pace: 97.5, prediction: 108.79970620159784, expected PPG: 106.9575, inflator: 0.9793276416353077
Team: Boston Celtics, Opponent: Golden State Warriors, season: 2022, Team ORtg: 114.4, Opp DRtg: 106.9, Pace: 97.5, prediction: 107.80822627846254, expected PPG: 107.88375, inflator: 1.0008490113711797
Team: Boston Celtics, Opponent: Golden State Warriors, season: 2022, Team ORtg: 114.4, Opp DRtg: 106.9, Pace: 97.5, prediction: 107.92363982637463, expected PPG: 107.88375, inflator: 0.9995515714900237
Team: Golden State Warriors, Opponent: Boston Celtics, season: 2022, Team ORtg: 112.5, Opp DRtg: 106.9, Pace: 97.5, prediction: 113.27931983827018, expected PPG: 106.9575, inflator: 0.930407408546838
```

Evaluation Results (Testing):

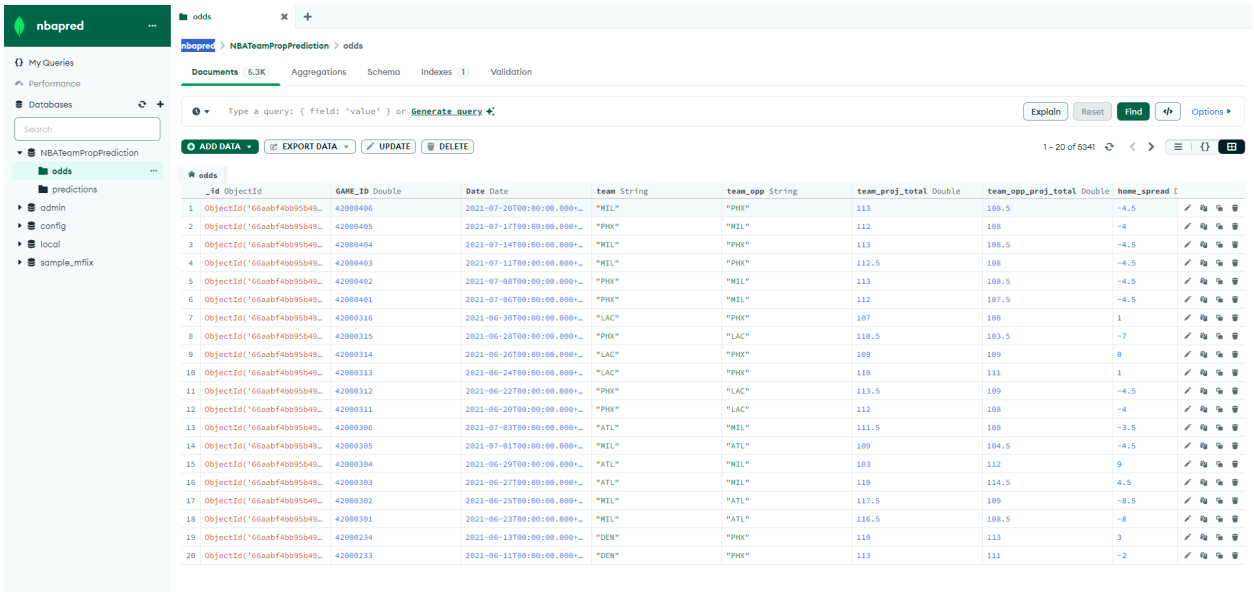


MongoDB Cloud Database:

Predictions Database:

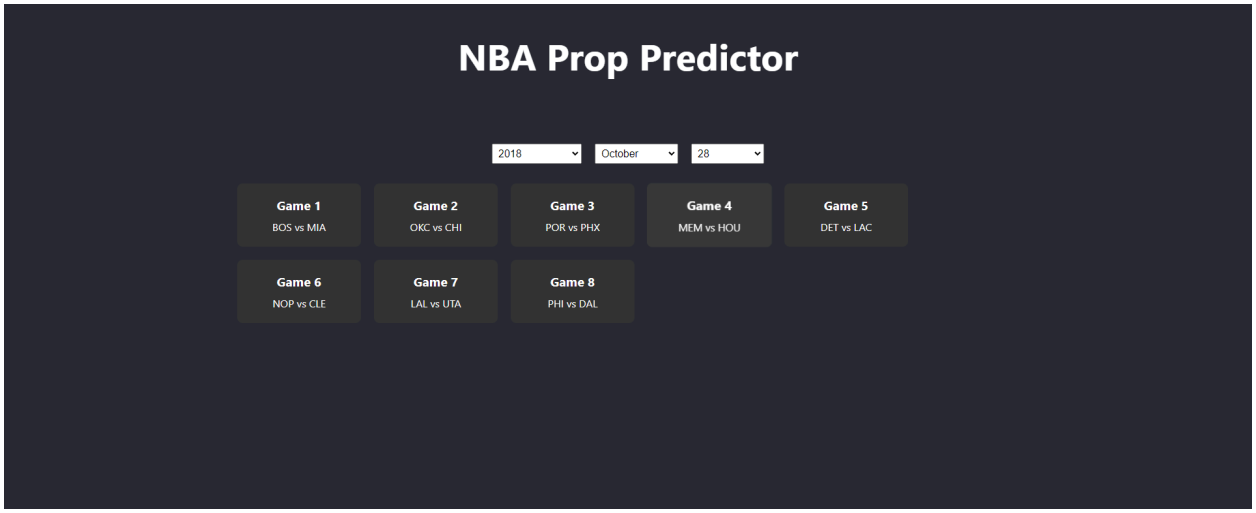


Odds Database:

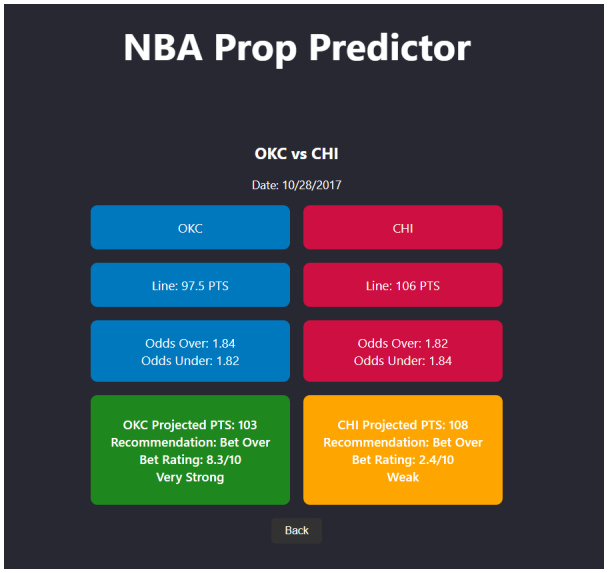


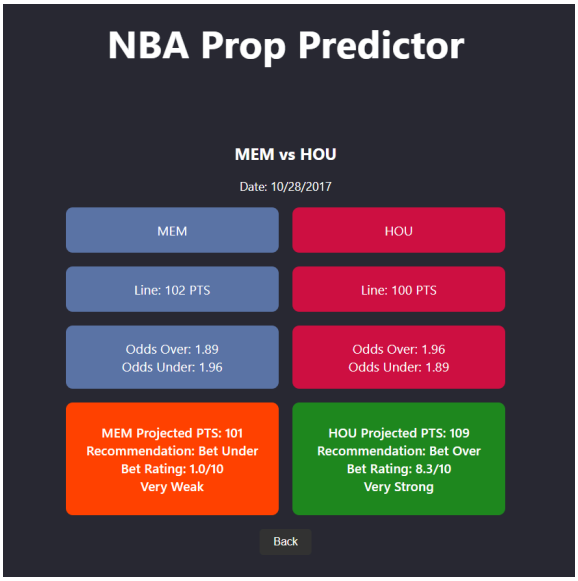
React Web App:

Page 1:



Page 2:





References

https://accuribet.win/bets

https://github.com/kyleskom/NBA-Machine-Learning-Sports-Betting/tree/master

https://github.com/NBA-Betting

https://www.nbastuffer.com/sports-data/

https://the-odds-api.com/liveapi/guides/v4/#overview

Stephen R. Schach - Object-Oriented and Classical Software Engineering, 8th Edition

https://github.com/shaneshamku/NBATeamPropPrediction/tree/main