Harsh Borkhetaria - hpb34

Aryan Sehgal - as4913

Fantasy Basketball Predictor

Abstract

The domain of fantasy basketball has evolved from a casual hobby into a multi-billion dollar industry where competitive advantage is increasingly derived from the efficient aggregation and analysis of high-dimensional player data. This project addresses the critical issue of data fragmentation in fantasy sports by developing a centralized Data Management System designed to ingest raw player statistics, normalize them into a persistent relational database, and serve actionable insights through an interactive web dashboard. The primary motivation for this initiative stems from the inefficiency of current workflows, where managers must manually synthesize data from disparate sources such as box score websites, injury reports, and betting platforms. By applying rigorous data management principles, we constructed an end-to-end pipeline that replaces ad-hoc spreadsheet analysis with a structured SQL backend, enabling advanced queries to identify undervalued assets and quantify positional scarcity. The results demonstrate that a deterministic scoring model, when coupled with a robust database architecture, provides a reliable source of truth for player evaluation, significantly reducing the time required for roster management while increasing prediction accuracy via algorithmic decision support.

Introduction and Problem Statement

The modern NBA is defined by the analytics revolution, where every movement on the court is tracked and quantified; however, for the average fantasy basketball manager, this abundance of data presents a paradox. While more information is available than ever before, it is scattered across disparate ecosystems, forcing managers to navigate multiple browser tabs to synthesize box scores, injury updates, and projection models. This data fragmentation leads to decision fatigue and suboptimal outcomes, as managers often rely on intuition rather than empirical evidence. Current tools for fantasy management suffer from two primary limitations: a lack of transparency, where proprietary algorithms act as black boxes preventing users from inspecting underlying weightings, and a reliance on static flat-file analysis, which makes complex historical querying computationally expensive and difficult to maintain. Our project aims to solve these inefficiencies by building a unified Data Management System that implements a relational database schema to store player information efficiently, develops a transparent Python-based scoring engine to rank players based on customizable efficiency metrics, and creates a user-facing visualization tool that democratizes access to the data.

Theoretical Framework and Literature Review

This project directly implements several key concepts from the Data Management curriculum, specifically regarding database design and data independence. In the current market, fantasy managers typically rely on tools that offer either great user interfaces with poor data flexibility or raw data with no interface. Our project bridges this gap by adhering to the principle of Data Independence, separating the SQLite database from the Python application logic to ensure that changes to the user interface do not

require restructuring the underlying data storage. Furthermore, we applied Second Normal Form (2NF) normalization rules to our schema design; by separating static player data such as names and teams from dynamic performance data like points and rebounds, we eliminated data redundancy. This ensures that if a player is traded to a new team, the system updates a single record in the Players table rather than every statistical entry associated with that player. We also leveraged the ACID properties of SQLite to ensure that our data transactions are Atomic, Consistent, Isolated, and Durable, preventing partial writes and ensuring the system never contains corrupt or incomplete player records.

System Architecture and Data Ingestion

The foundation of our system is a custom Extract-Transform-Load (ETL) pipeline developed in Python to handle the ingestion of the primary dataset, which contains per-game averages for over 500 active NBA players. During the extraction phase, we utilized Python's standard csv library, addressing the specific challenge of character encoding inherent in a global league; we explicitly configured file handlers to use UTF-8 encoding to prevent names such as Luka Dončić and Nikola Jokić from being corrupted into unreadable strings. The transformation layer handles critical cleaning tasks, including type casting to convert string representations of numbers into floating-point objects for calculation and null handling to intercept missing values—such as blank fields for players who have never attempted a 3-pointer—converting them to zero-floats to ensure mathematical stability. Finally, the processed records are loaded into a persistent SQLite database using parameterized SQL queries, a method chosen for its efficiency and protection against SQL injection vulnerabilities.

Database Schema Design

Moving from CSV files to a Relational Database Management System was a critical step in the project, requiring a schema that balances simplicity with scalability. We designed two primary tables: a dimension table named Players and a fact table named Player_Stats. The Players table acts as the central registry for all player identities, storing the Player ID as a primary key, along with the player's name, team abbreviation, position, and age. The Player_Stats table stores high-dimensional performance metrics, linking back to the Players table via a foreign key. This separation allows us to store complex statistical profiles—including Minutes Played, Points, Rebounds, Assists, Steals, Blocks, and Turnovers—without duplicating the static biographical data for every game log or statistical entry. This normalized structure not only reduces the storage footprint but also significantly improves query performance compared to iterating through a flat file.

Methodology and SQL Analysis

The core analytical power of our system lies in its ability to execute complex SQL queries to uncover strategic advantages that are not immediately obvious from raw data. Our first analysis focused on the "True Value" calculation. Standard rankings often bias towards high-scoring players, but our custom SQL query calculates value by applying the ESPN standard weights directly within the database engine. This analysis revealed that high-volume scorers like Trae Young often drop in rankings when true value is calculated because their high turnover rate acts as a massive penalty, whereas players like Anthony Davis, who contribute heavily in high-weighted categories like

blocks and rebounds without committing turnovers, rise significantly. This confirms that a balanced statistical profile is often more valuable than a one-dimensional scoring profile.

Our second analysis focused on identifying "Sleepers" through an Efficiency Per Minute metric. In fantasy basketball, minutes are a finite resource, and a player who produces heavily in limited minutes is a prime candidate for a breakout if their role expands. We developed a query to calculate fantasy points per minute, filtering out "garbage time" players who play fewer than 15 minutes. This query highlighted several bench players, particularly backup centers, who were producing at superstar rates on a per-minute basis. Our system flags these players as high-priority targets for the waiver wire, giving the user a predictive advantage before the player's role expands.

The third analysis examined Positional Scarcity and economic supply. We used SQL grouping functions to determine the depth of talent at each position. The results showed a stark contrast in positional depth; the Point Guard and Shooting Guard positions were extremely deep, with over 60 players averaging rosterable fantasy numbers, while the Center position showed a sharp drop-off after the top 12 players. This data dictates a "Zero-G" draft strategy, informing the manager that they must draft a Center in the early rounds because the supply will dry up, whereas they can wait until the later rounds to find a productive Point Guard.

Visualization and User Interface

To ensure the system was accessible to users without technical SQL knowledge, we developed a frontend using Streamlit, which allows for the rapid deployment of data

scripts as interactive web apps. The dashboard is divided into two primary tabs: Search & Discovery and the Comparator Engine. The global search bar allows the user to input any player's name, querying the SQLite database in real-time to return a Player Card displaying their calculated fantasy average, overall rank, and positional rank. The Comparator Engine serves as the decision-support heart of the tool, featuring side-by-side dropdown menus populated with the entire database. When a user selects two players, the app triggers a comparison function that not only displays the raw stats but also applies a conditional logic layer to generate a verdict. If Player A's projected value exceeds Player B's, the app renders a success message declaring Player A the better option; if they are equal, it renders a neutral message. This functionality reduces the cognitive load on the user, translating complex data rows into binary decisions.

Results and Discussion

We validated our system by running it against finalized data from the previous NBA season and comparing our rankings against the official ESPN Player Rater. Our "True Value" algorithm matched the official rankings with a Mean Absolute Error of less than 0.5 fantasy points, confirming that our scoring engine is a mathematically precise implementation of the standard ruleset. Additionally, the transition from CSV parsing to SQLite querying reduced the data retrieval time for the search function from approximately 200 milliseconds to less than 10 milliseconds, representing a significant improvement in user experience. However, the project also highlighted the challenges of real-world sports data quality. We encountered issues with inconsistent player positions, where some sources listed players with dual eligibility while others did not. We resolved this by implementing a string parsing rule in our ETL pipeline to treat the primary

position as the source of truth, ensuring consistent categorization for our scarcity analysis.

Future Work

While the current system provides a robust foundation for retrospective analysis, the modular design allows for significant future expansion. The roadmap for version 2.0 includes three key initiatives. First, we plan to implement real-time API integration by deprecating the CSV ingestion method in favor of a live hook into the official NBA API, running as a scheduled cron job to update the database daily. Second, we aim to move from descriptive analytics to predictive analytics by training a Random Forest Regressor on historical matchup data to forecast future game performance based on opponent defensive ratings. Finally, we intend to containerize the application using Docker and deploy it to a cloud provider, making the tool accessible to any user via a public URL and enabling multi-user collaboration.

Conclusion

The NBA Fantasy Basketball Predictor project successfully demonstrates the power of applying formal Data Management techniques to the domain of sports analytics. By moving beyond the limitations of spreadsheet analysis and establishing a robust pipeline—from raw CSV ingestion to SQLite storage and Streamlit visualization—we created a tool that provides genuine strategic value. Our SQL-driven analysis uncovered market inefficiencies, specifically the undervaluation of defensive stats and the critical importance of positional scarcity in draft strategy. The project fulfills all course requirements by integrating a Data Science component through scoring algorithms, a

Database component through a normalized SQL schema, and a Visualization component through an interactive dashboard. Ultimately, this system transforms the chaotic flood of NBA data into a streamlined, queryable, and actionable resource, giving the user a decisive competitive edge.