Máster en Basket Data Analytics & Sport Management

AssistAI: All-In-One Basket Analytics App

Trabajo fin de máster presentado por:	Mateo Peciña Marqueta
Director/a:	Francisco Camba Rodríguez
Fecha:	15/08/2025

En colaboración con:

Resumen / Abstract

Este Trabajo de Fin de Máster (TFM) presenta AssistAI, una herramienta web integral todo en uno para entrenadores de baloncesto, que centraliza el análisis de datos avanzados. La plataforma utiliza estadísticas detalladas de los dos últimos años de la Euroliga, obtenidas de HackAStat.

Las funcionalidades principales de AssistAI incluyen:

- **Chatbot Inteligente:** Responde preguntas sobre datos, explica términos estadísticos y genera informes de partidos a partir de URLs, todo en lenguaje natural.
- **Informes y Comparativas:** Ofrece visualizaciones avanzadas para analizar el rendimiento de jugadores y equipos.
- **Buscador de Jugadores Similares:** Identifica jugadores con perfiles estadísticos parecidos, con opciones de filtrado y ponderación de métricas.
- **Motor Predictivo de Quintetos:** Estima el rendimiento colectivo (Net Rating) de un quinteto seleccionado, basándose en estadísticas individuales.

Construida con PostgreSQL, FastAPI y Streamlit con una estructura de capas, AssistAI proporciona una interfaz intuitiva que transforma datos complejos en inteligencia accionable, facilitando la toma de decisiones en el baloncesto profesional.

This Master's Thesis (TFM) presents AssistAI, a comprehensive all-in-one web tool for basketball coaches that centralizes advanced data analysis. The platform uses detailed statistics from the last two years of the Euroleague, obtained from HackAStat.

AssistAI's key features include:

- **Intelligent Chatbot**: Answers questions about data, explains statistical terms, and generates game reports from URLs, all in natural language.
- **Reports and Comparisons**: Offers advanced visualizations to analyze player and team performance.
- **Similar Player Finder**: Identifies players with similar statistical profiles, with filtering and metric weighting options.
- **Predictive Starting Five Engine**: Estimates the collective performance (Net Rating) of a selected starting five, based on individual statistics.

Built with PostgreSQL, FastAPI, and Streamlit with a layered structure, AssistAI provides an intuitive interface that transforms complex data into actionable intelligence, facilitating decision-making in professional basketball.

Palabras clave / Keywords: Euroliga, multi-agente, FastAPI, Streamlit, Machine Learning / Euroleague, multi-agentic, FastAPI, Streamlit, Machine Learning

Índice de contenidos

1. Introducción

Este Trabajo de Fin de Máster tiene como objetivo el desarrollo de una herramienta web integral dirigida a entrenadores de baloncesto, que unifique distintas funcionalidades basadas en el análisis de datos avanzados. La plataforma estará alimentada por datos individuales y colectivos de jugadores y equipos de ligas como la Euroliga, extraídos desde APIs públicas. Las funcionalidades principales incluirán:

1. Un chatbot capaz de responder en lenguaje natural a preguntas basadas en datos, dar explicación a los términos estadísticos de baloncesto y generar un informe de las estadísticas de un partido buscando dar explicaciones al resultado final basándose en estas.
2. Módulo para generar visualizaciones avanzadas y comparativas de jugadores y equipos y contexto de liga.
3. Buscador de jugadores similares
4. Motor predictivo que estime el rendimiento de un quinteto seleccionado.

Todo esto se articulará a través de un backend basado en FastAPI y un frontend con Streamlit.

1.1. Motivación

En el dinámico y cada vez más competitivo mundo del baloncesto profesional, la toma de decisiones informadas es crucial para el éxito. Entrenadores, analistas y directivos buscan constantemente ventajas que les permitan optimizar el rendimiento de sus equipos y jugadores. Tradicionalmente, el análisis se basaba en la observación y la experiencia, pero la irrupción de los datos avanzados ha revolucionado esta perspectiva. Hoy en día, la capacidad de procesar, interpretar y aplicar grandes volúmenes de información estadística es un diferenciador clave.

Sin embargo, a pesar de la abundancia de datos disponibles, a menudo se encuentran dispersos en diversas fuentes, con formatos inconsistentes y sin herramientas que permitan una explotación sencilla y eficaz. Los entrenadores, en particular, necesitan soluciones que les ofrezcan acceso rápido a insights relevantes, sin requerir conocimientos profundos de programación o estadística avanzada. La motivación principal detrás de este Trabajo de Fin de Máster (TFM) surge de esta necesidad: desarrollar una herramienta que democratice el acceso

a la analítica avanzada en baloncesto, proporcionando una plataforma unificada y fácil de usar que transforme datos crudos en inteligencia accionable.

Mi objetivo es cerrar la brecha entre la complejidad de los datos avanzados y la simplicidad que requiere un entorno de toma de decisiones rápida como el baloncesto. Quiero que esta solución no solo sea una base de datos, sino un asistente inteligente que responda preguntas, visualice tendencias y prediga escenarios. Todo ello en un lenguaje accesible para el usuario final.

1.2. ¿Qué aporta esta solución?

Esta plataforma integral aporta un valor significativo al ecosistema del baloncesto a través de varias funcionalidades clave que unifican y simplifican el análisis de datos avanzados:

- Acceso Unificado a Datos Avanzados: Centraliza información detallada de jugadores y equipos de la Euroliga, eliminando la necesidad de consultar múltiples fuentes y lidiar con formatos dispares. Esto ahorra tiempo y reduce la fricción en el proceso de análisis.
- Interacción en Lenguaje Natural (Chatbot): Permite a los usuarios realizar consultas complejas sobre estadísticas y conceptos de baloncesto utilizando lenguaje natural. El chatbot no solo recupera datos específicos, sino que también explica términos técnicos y genera informes de partidos a partir de URLs, facilitando la comprensión y el aprendizaje.
- Visualizaciones Avanzadas y Comparativas: Ofrece dashboards interactivos y personalizables para comparar el rendimiento de jugadores y equipos. Estas visualizaciones van más allá de las estadísticas básicas, incorporando métricas avanzadas que proporcionan una comprensión más profunda del impacto en el juego.
- Buscador de Jugadores Similares: Implementa un algoritmo de similitud que ayuda a identificar jugadores con perfiles estadísticos parecidos, una funcionalidad invaluable para el scouting, la planificación de plantillas o el análisis de rivales. Permite ajustar la importancia de diferentes métricas y filtrar por rol.
- Motor Predictivo de Quintetos: Proporciona una herramienta innovadora para estimar el rendimiento colectivo (Net Rating) de cualquier quinteto seleccionado, basándose

en las estadísticas individuales de sus integrantes. Esto permite a los entrenadores simular y optimizar combinaciones de jugadores antes de que salten a la cancha.

- Interfaz de Usuario Intuitiva: A través de un frontend desarrollado con Streamlit, la plataforma es accesible y fácil de usar, incluso para aquellos sin experiencia previa en análisis de datos. La navegación es fluida y las funcionalidades están claramente organizadas.

En esencia, esta solución transforma un mar de datos en una herramienta de inteligencia estratégica, permitiendo a los profesionales del baloncesto tomar decisiones más rápidas, precisas y basadas en evidencia.

1.3. Objetivos

Para lograr la visión de una plataforma integral de análisis de datos en baloncesto, se establecieron los siguientes objetivos específicos:

- **Desarrollar un Sistema de Extracción y Almacenamiento de Datos Robusto**:
  - Identificar y seleccionar fuentes de datos fiables y completas de estadísticas de baloncesto.
  - Diseñar un esquema de base de datos relacional (PostgreSQL) optimizado para el almacenamiento eficiente de estadísticas de jugadores, equipos y temporadas.
  - Implementar procesos ETL (Extract, Transform, Load) para la recolección, limpieza, transformación y carga automatizada de datos en la base de datos.
- **Construir un Backend API RESTful con FastAPI**:
  - Diseñar y desarrollar endpoints API para cada funcionalidad principal (chat, predicción de rendimiento, informes de jugadores/equipos, búsqueda de similares, acceso a datos crudos).
  - Integrar un sistema multi-agente (CrewAI) en el endpoint de chat para permitir la interacción en lenguaje natural, la explicación de términos estadísticos y la generación de informes de partidos a partir de URLs.
  - Implementar un modelo de Machine Learning para la predicción del Net Rating de quintetos, asegurando su integración eficiente en el backend.

  - Desarrollar la lógica de negocio para la búsqueda de jugadores similares, incluyendo la ponderación de métricas y filtros por rol.
- **Crear un Frontend Interactivo y Amigable con Streamlit**:
  - Diseñar una interfaz de usuario intuitiva y visualmente atractiva que facilite la interacción con todas las funcionalidades del backend.
  - Implementar dashboards dinámicos para la visualización y comparación de estadísticas de jugadores y equipos, utilizando gráficos avanzados.
  - Asegurar una navegación sencilla y coherente entre las diferentes secciones de la aplicación.
- **Garantizar la Escalabilidad y Mantenibilidad de la Solución**:
  - Adoptar una arquitectura por capas que promueva la modularidad y la separación de responsabilidades.
  - Utilizar tecnologías y frameworks modernos y bien documentados (FastAPI, Streamlit, PostgreSQL, CrewAI).
  - Implementar buenas prácticas de codificación y documentación para facilitar futuras mejoras y el mantenimiento.
- **Validar la Utilidad y Precisión de las Funcionalidades**:
  - Realizar pruebas exhaustivas de cada módulo para asegurar la correcta recuperación, procesamiento y presentación de los datos.
  - Evaluar la precisión del modelo predictivo y la relevancia de los resultados de búsqueda de jugadores similares.

Estos objetivos, abordados de manera secuencial y modular, han guiado el desarrollo de la plataforma, buscando ofrecer una herramienta de alto valor para la comunidad del baloncesto.

2. Estructura principal

El proyecto ha sido concebido y desarrollado siguiendo una arquitectura por capas, un enfoque que promueve la modularidad, la separación de responsabilidades y la escalabilidad. Esta estructura facilita el desarrollo, la depuración y el mantenimiento del sistema, permitiendo que cada componente se enfoque en una tarea específica sin interferir con las demás.

2.1. Estructura por capas

La solución se articula en tres capas principales:

- **Capa de Base de Datos (BD)**: Es el cimiento donde reside toda la información. Su función principal es el almacenamiento persistente y la gestión de los datos de baloncesto. Esta capa es la única que interactúa directamente con el sistema de archivos o los servicios de bases de datos.
- **Capa de Backend**: Actúa como el cerebro de la aplicación. Es la lógica de negocio que procesa las solicitudes del frontend, interactúa con la base de datos para obtener o manipular información, y ejecuta algoritmos complejos (como el modelo predictivo o la búsqueda de similitud). El backend nunca interactúa directamente con el frontend; lo hace a través de una API.
- **Capa de Frontend**: Es la interfaz de usuario, lo que el usuario ve y con lo que interactúa. Su responsabilidad es presentar la información de manera clara y atractiva, y capturar las interacciones del usuario para enviarlas al backend. El frontend nunca interactúa directamente con la base de datos; todas sus solicitudes de datos o procesamiento se canalizan a través del backend.

Esta jerarquía estricta asegura que las dependencias fluyan en una única dirección, limitando a las capas a poder interactuar únicamente con aquellas que tiene más próximas (eliminando así la posibilidad de que el frontend interactúe con la BD, para ello, tendrá que pasar por el backend) lo que simplifica la depuración, permite la sustitución de componentes (por ejemplo, cambiar la base de datos sin afectar el frontend) y facilita el trabajo en equipo.

Además, quise separar del resto de código los valores de configuración que se toman en cualquiera de las capas, como pueden ser el nombre, usuario y contraseñas de las Bases de

Datos u otro tipo de configuración. Para ello, crearemos un fichero con las constantes a las que después llamaremos para recuperar el valor de estas configuraciones para su uso. Para el caso de las contraseñas y valores sensibles, estos irán separados del resto de configuración, para lo cual se usará un fichero `.env`.

2.2. Capa BD

La capa de base de datos es fundamental para la persistencia y organización de la vasta cantidad de estadísticas de baloncesto. A lo largo del desarrollo, se exploraron diferentes fuentes de datos y se tomaron decisiones clave para optimizar la calidad y la granularidad de la información.

2.2.1. Fuentes de datos y decisiones

Inicialmente, la idea era construir una base de datos lo más amplia posible, abarcando diversas ligas y utilizando APIs deportivas generales.

- **Fase Inicial (Basic - SportRadar API)**:
    - **Fuente**: Tras un estudio de mercado de APIs que cumplían los requisitos pedidos, la primera aproximación se realizó utilizando la API de SportRadar. Esta API ofrecía acceso a datos de múltiples ligas, lo que parecía prometedor para una solución integral.
    - **Tecnología**: Se utilizó PostgreSQL como sistema de gestión de bases de datos relacionales para almacenar la información extraída.
    - **Proceso**: Se implementaron scripts de extracción (ubicados en `src/etl/basic`) para recolectar datos individuales y colectivos de jugadores y equipos. Tras la extracción, se realizaba un preprocesamiento básico para limpiar y estructurar los datos antes de su carga en la base de datos.
    - **Decisión**: Aunque SportRadar ofrecía una cobertura amplia, la prueba gratuita tenía limitaciones significativas en cuanto al número de peticiones. Rápidamente, se agotó el límite, lo que hizo inviable continuar con esta fuente para un proyecto de la envergadura deseada. Esto llevó a la necesidad de reevaluar la estrategia de adquisición de datos.

- **Fase Final (Advanced - HackAStat)**:
    - **Fuente**: Tras la limitación con SportRadar, se decidió enfocar el proyecto en una única competición de alto nivel para asegurar la profundidad y calidad de los datos. La Euroliga fue la elección natural. Se realizó un nuevo estudio de mercado y se identificó `hackastat.eu` como una fuente excepcional de estadísticas avanzadas de la Euroliga. Esta plataforma ofrecía una granularidad de datos mucho mayor, incluyendo métricas avanzadas que son cruciales para un análisis profundo en baloncesto.
    - **Tecnología**: Se mantuvo PostgreSQL, dada su robustez y flexibilidad para manejar esquemas de datos complejos.
    - **Proceso**: Se diseñó un nuevo esquema de base de datos, más detallado, para acomodar la riqueza de las estadísticas de `hackastat.eu`. Los scripts ETL (ubicados en `src/etl/advanced`) se adaptaron para extraer, transformar y cargar estos nuevos datos. Se priorizó la inclusión de las temporadas 2023/24 y 2024/25, consideradas suficientes para la primera versión de la aplicación y para demostrar las capacidades analíticas.
    - **Decisión**: Esta transición a `hackastat.eu` fue una decisión estratégica clave. Aunque se redujo el alcance a una sola liga, la calidad y profundidad de los datos obtenidos compensaron con creces la limitación, permitiendo análisis mucho más sofisticados y relevantes para el contexto del baloncesto profesional.

2.2.2. ETL

El proceso ETL (Extract, Transform, Load) es el corazón de la capa de datos, encargado de poblar y mantener la base de datos con información actualizada y estructurada.

2.2.2.1. Basic

En la fase inicial del proyecto, el proceso ETL para la base de datos “Basic” se centró en la recolección de datos de la API de SportRadar.

- **Extracción (Extract)**: Se desarrollaron scripts en Python que interactuaban con la API de SportRadar. Gracias a la librería `requests`, estos scripts realizaban llamadas a los

diferentes endpoints de la API para obtener información sobre ligas, equipos, jugadores y sus estadísticas básicas (puntos, rebotes, asistencias, etc.). La recolección se diseñó para ser lo más exhaustiva posible dentro de las limitaciones de la API. Para ello, desde el fichero de configuración podremos seleccionar de qué ligas queremos recopilar los datos (de entre todas las disponibles en la API usada) y así, automáticamente el proceso comenzará buscando y recopilando los datos estadísticos de todos los partidos de las temporadas de las ligas elegidas.

- **Transformación (Transform)**: Una vez extraídos los datos, se aplica una serie de transformaciones. Esto incluye:
    - **Limpieza**: Manejo de valores nulos, corrección de tipos de datos (por ejemplo, hay que asegurar que las estadísticas numéricas se almacenaran como números).
    - **Normalización**: Estructuración de los datos en un formato tabular adecuado para una base de datos relacional. Esto implicaba aplanar estructuras JSON anidadas y crear relaciones entre las diferentes entidades (jugadores, equipos, temporadas, estadísticas).
    - **Cálculo de Métricas Derivadas:** Aunque la API de SportRadar proporcionaba estadísticas básicas, se calculan algunas estadísticas más avanzadas que podemos extraer a partir de las básicas, como pueden ser el número de posesiones, el porcentaje de rebotes, asistencias entre otras.
- **Carga (Load)**: Los datos transformados se cargaban en una base de datos PostgreSQL. Se utilizaban librerías como `psycopg2` en Python para establecer la conexión con la base de datos y ejecutar sentencias SQL (INSERT, UPDATE) para poblar las tablas. El esquema de la base de datos se diseñó para reflejar las entidades principales: `competitions`, `seasons`, `teams`, `sport_events`, `team_statistics`, `players`, y `player_statistics` y `season_player_statistics`. Esta última tabla contiene únicamente datos calculados a partir de sus estadísticas normales calculamos las de la temporada.

Este proceso, aunque funcional, se vio limitado por las restricciones de la API de SportRadar, lo que eventualmente llevó a la necesidad de buscar una fuente de datos más adecuada para el alcance del TFM.

A pesar de ello, nos ofrece cosas que la solución elegida no lo hace, como la facilidad de realizar el proceso de ETL de manera totalmente automática, sin necesidad de realizar descargas a mano como ahora veremos. Además, permite la inclusión de muchas ligas, ya que la API cuenta con datos de todo el mundo e incluso de ligas (o torneos) de equipos de formación (como por ejemplo las selecciones U16).

2.2.2.2. Advanced

Tras desechar la opción previamente explicada, tocaba realizar un nuevo estudio de mercado. Para ello, cambiamos el enfoque, ya que en el primero, buscamos una API que fuera capaz de darnos datos sobre muchas ligas, sin importar la complejidad de estos datos. Ahora, le damos un poco la vuelta y buscamos recopilar datos avanzados, aunque sea solo de una única liga, la cual sería la Euroliga, ya que a modo de muestra es más que suficiente contar con una liga, siendo más importante contar con unos datos mejores y más avanzados.

La transición a `hackastat.eu` para la base de datos "Advanced" implicó una revisión y mejora significativa del proceso ETL, dado el mayor volumen y la complejidad de las estadísticas disponibles.

- **Extracción (Extract)**: Los scripts de extracción (`src/etl/advanced`) se reescribieron para interactuar con la información disponible en `hackastat.eu`. Aunque al no tratarse de API en el sentido tradicional, se tuvo que descargar los datos a mano de la propia web en formato Excel que posteriormente transformamos a CSV. Tras esto, se desarrolló un método para parsear y extraer los datos de las páginas relevantes de la Euroliga para las temporadas 2023/24 y 2024/25. Esto incluía estadísticas mucho más detalladas, como ratings ofensivos/defensivos, porcentajes de tiro efectivos, uso de posesión, etc
- **Transformación (Transform)**: Esta fase fue mucho más intensiva debido a la riqueza de los datos:
    - **Limpieza y Estandarización**: Se manejaron inconsistencias en los nombres de jugadores/equipos, se convirtieron cadenas de texto a tipos numéricos

adecuados (especialmente para porcentajes y promedios), y se gestionaron valores faltantes. Además, se eliminan aquellos jugadores cuyo aporte haya sido muy bajo, esto es debido a que desvirtuaban varios cálculos y hacían que el chat contestara algunas cosas que no tenían sentido.

  - **Cálculo de Métricas Avanzadas**: A diferencia de lo que sucedía en el caso anterior, ahora ya contábamos con suficientes métricas para poder dar una visión global y realista tanto de los jugadores como de los equipos, por lo que no tuvimos que calcular métricas extras.
  - **Esquema de BD enriquecido**: El esquema de la base de datos PostgreSQL se amplió considerablemente para albergar todas estas nuevas métricas. Se crearon tablas como `seasons`, `teams`, `players`, `team_stats`, y `player_stats`, con una gran cantidad de columnas para cada estadística avanzada.

- **Carga (Load)**: De manera prácticamente igual al caso anterior, los datos transformados se cargaban en la nueva base de datos PostgreSQL. Se utilizaron sentencias `INSERT`, ya que al tener que descargar los archivos a mano, perdíamos la opción de actualizar automáticamente las tablas, por lo que no fueron necesarias sentencias `UPDATE`. La robustez de PostgreSQL permitió manejar la complejidad de este nuevo conjunto de datos de manera eficiente.

Este proceso ETL avanzado fue fundamental para construir una base de datos rica y precisa, que sirviera como el motor de todas las funcionalidades analíticas de la plataforma, asentando las bases de lo que sería nuestra aplicación.

2.2.3. Fragmentos de código

`src/etl/basic/api/sportradar_client.py`: fichero que contiene al cliente encargado de hacer las peticiones a la API de SportRadar (como se puede apreciar, existe delay en las peticiones, ya que la versión de prueba de la API te limita las peticiones)

```python
import requests
import time

class SportRadarClient:
    def __init__(self, api_key, base_url, locale):
        self.api_key = api_key
        self.base_url = base_url
        self.locale = locale
        self.session = requests.Session()

    def _get_request(self, endpoint, params=None):
        """Método privado para manejar las solicitudes GET, errores y reintentos."""
```

```
        url = f"{self.base_url}/{self.locale}/{endpoint}"

        full_params = {"api_key": self.api_key}
        if params:
            full_params.update(params)

        retries = 5
        delay = 30

        for i in range(retries):
            try:
                response = self.session.get(url, params=full_params, headers={"accept":
"application/json"})

                if response.status_code == 429:
                    print(f"Error 429: Too Many Requests. Reintentando en {delay} segundos...")
                    time.sleep(delay)
                    delay *= 2
                    continue

                response.raise_for_status()
                return response.json()

            except requests.exceptions.RequestException as e:
                print(f"Error al llamar a la API para el endpoint {endpoint}")
                return None

        print(f"Fallo en la solicitud después de {retries} reintentos.")
        return None

    def get_competition_seasons(self, urn_competition):
        """Obtiene todas las temporadas para una competición."""
        endpoint = f"competitions/{urn_competition}/seasons"
        return self._get_request(endpoint)

    def get_season_summaries(self, urn_season, limit=200, offset=0):
        """Obtiene resúmenes de eventos deportivos para una temporada con paginación."""
        endpoint = f"seasons/{urn_season}/summaries"
        params = {"limit": limit, "offset": offset}
        return self._get_request(endpoint, params)

    def get_sport_event_summary(self, urn_sport_event):
        """Obtiene el resumen detallado de un evento deportivo."""
        endpoint = f"sport_events/{urn_sport_event}/summary"
        return self._get_request(endpoint)

    def get_season_competitor_statistics(self, urn_season, urn_competitor):
        """Obtiene las estadísticas de un equipo y sus jugadores para una temporada."""
        endpoint = f"seasons/{urn_season}/competitors/{urn_competitor}/statistics"
        return self._get_request(endpoint)

    def get_season_standings(self, urn_season, live=False):
        """Obtiene la clasificación de una temporada."""
        endpoint = f"seasons/{urn_season}/standings"
        params = {"live": live}
        return self._get_request(endpoint, params)
```

`src/etl/advanced/main.py`: fichero con toda la lógica de creación de la BD “Advanced”, aquí podemos apreciar la cantidad de métricas avanzadas con las que contará esta nueva BD.

```
import pandas as pd
import psycopg2
import os
from config import ADVANCED_DB_CONFIG, CSV_FILES
```

```
def create_advanced_database_and_tables():
    conn = None
    try:
        # Conexión sin especificar la base de datos para crear una nueva
        conn = psycopg2.connect(
            user=ADVANCED_DB_CONFIG['user'],
            password=ADVANCED_DB_CONFIG['password'],
            host=ADVANCED_DB_CONFIG['host'],
            dbname='postgres' # Conectar a la base de datos por defecto
        )
        conn.autocommit = True
        cur = conn.cursor()

        # Crear la base de datos si no existe
        db_name = ADVANCED_DB_CONFIG['dbname']
        cur.execute(f"SELECT 1 FROM pg_database WHERE datname='{db_name}'")
        if not cur.fetchone():
            cur.execute(f"CREATE DATABASE {db_name};")
            print(f"Base de datos '{db_name}' creada.")
        else:
            print(f"Base de datos '{db_name}' ya existe.")

        # Cerrar la conexión inicial
        cur.close()
        conn.close()

        # Conectar a la nueva base de datos y crear tablas
        conn = psycopg2.connect(**ADVANCED_DB_CONFIG)
        conn.autocommit = True
        cur = conn.cursor()

        # Definición de las tablas
        table_commands = [
            """
            CREATE TABLE IF NOT EXISTS seasons (
                season_id VARCHAR(10) PRIMARY KEY,
                season_name VARCHAR(50)
            )
            """,
            """
            CREATE TABLE IF NOT EXISTS teams (
                tm_name VARCHAR(255) PRIMARY KEY,
                season_id VARCHAR(10) REFERENCES seasons(season_id)
            )
            """,
            """
            CREATE TABLE IF NOT EXISTS players (
                name VARCHAR(255) PRIMARY KEY,
                role VARCHAR(50),
                nat VARCHAR(50),
                height INTEGER,
                age INTEGER,
                tm_name VARCHAR(255) REFERENCES teams(tm_name),
                season_id VARCHAR(10) REFERENCES seasons(season_id)
            )
            """,
            """
            CREATE TABLE IF NOT EXISTS team_stats (
                id SERIAL PRIMARY KEY,
                tm_name VARCHAR(255) REFERENCES teams(tm_name),
                season_id VARCHAR(10) REFERENCES seasons(season_id),
                gp INTEGER,
                w INTEGER,
                l INTEGER,
                min FLOAT,
                pts FLOAT,
                two_ptm FLOAT,
                two_pta FLOAT,
                two_pt_pct FLOAT,
```

```
    three_ptm FLOAT,
    three_pta FLOAT,
    three_pt_pct FLOAT,
    fgm FLOAT,
    fga FLOAT,
    fg_pct FLOAT,
    ftm FLOAT,
    fta FLOAT,
    ft_pct FLOAT,
    or_rebounds FLOAT,
    dr_rebounds FLOAT,
    tr_rebounds FLOAT,
    ast FLOAT,
    tovers FLOAT,
    st FLOAT,
    blk FLOAT,
    blka FLOAT,
    pf FLOAT,
    df FLOAT,
    val FLOAT,
    plus_minus FLOAT,
    pace FLOAT,
    poss FLOAT,
    shooting_chances FLOAT,
    off_ppp FLOAT,
    def_ppp FLOAT,
    off_rtg FLOAT,
    def_rtg FLOAT,
    net_rtg FLOAT,
    efg_pct FLOAT,
    ts_pct FLOAT,
    rim_freq FLOAT,
    rim_pps FLOAT,
    paint_freq FLOAT,
    paint_pps FLOAT,
    mid_freq FLOAT,
    mid_pps FLOAT,
    c3_freq FLOAT,
    c3_pps FLOAT,
    l3_freq FLOAT,
    l3_pps FLOAT,
    ft_ratio FLOAT,
    to_pct FLOAT,
    lto_pct FLOAT,
    dto_pct FLOAT,
    ast_pct FLOAT,
    ast_pct_2p FLOAT,
    ast_pct_3p FLOAT,
    ast_pct_ft FLOAT,
    ast_ratio FLOAT,
    ast_to_ratio FLOAT,
    or_pct FLOAT,
    or_pct_after_2p FLOAT,
    or_pct_after_3p FLOAT,
    or_pct_after_ft FLOAT,
    dr_pct FLOAT,
    dr_pct_after_2p FLOAT,
    dr_pct_after_3p FLOAT,
    dr_pct_after_ft FLOAT,
    tr_pct FLOAT,
    st_pct FLOAT,
    blk_pct FLOAT,
    blk_pct_2p FLOAT,
    blk_pct_3p FLOAT,
    kills FLOAT,
    psf_freq FLOAT,
    dsf_freq FLOAT,
    sos FLOAT
)
""",
```

```
        """
        CREATE TABLE IF NOT EXISTS player_stats (
            id SERIAL PRIMARY KEY,
            name VARCHAR(255) REFERENCES players(name),
            tm_name VARCHAR(255) REFERENCES teams(tm_name),
            season_id VARCHAR(10) REFERENCES seasons(season_id),
            role VARCHAR(50),
            nat VARCHAR(50),
            height INTEGER,
            age INTEGER,
            gp INTEGER,
            w INTEGER,
            l INTEGER,
            w_pct FLOAT,
            min FLOAT,
            pts FLOAT,
            two_ptm FLOAT,
            two_pta FLOAT,
            two_pt_pct FLOAT,
            three_ptm FLOAT,
            three_pta FLOAT,
            three_pt_pct FLOAT,
            fgm FLOAT,
            fga FLOAT,
            fg_pct FLOAT,
            ftm FLOAT,
            fta FLOAT,
            ft_pct FLOAT,
            or_rebounds FLOAT,
            dr_rebounds FLOAT,
            tr_rebounds FLOAT,
            ast FLOAT,
            tovers FLOAT,
            st FLOAT,
            blk FLOAT,
            blka FLOAT,
            pf FLOAT,
            df FLOAT,
            val FLOAT,
            plus_minus FLOAT,
            poss FLOAT,
            usg_pct FLOAT,
            ppp FLOAT,
            off_rtg_on FLOAT,
            def_rtg_on FLOAT,
            net_rtg_on FLOAT,
            ind_off_rtg FLOAT,
            ind_def_rtg FLOAT,
            ind_net_rtg FLOAT,
            efg_pct FLOAT,
            ts_pct FLOAT,
            rim_freq FLOAT,
            rim_pps FLOAT,
            paint_freq FLOAT,
            paint_pps FLOAT,
            mid_freq FLOAT,
            mid_pps FLOAT,
            c3_freq FLOAT,
            c3_pps FLOAT,
            l3_freq FLOAT,
            l3_pps FLOAT,
            ft_ratio FLOAT,
            to_pct FLOAT,
            lto_pct FLOAT,
            dto_pct FLOAT,
            ast_pct FLOAT,
            ast_pct_2p FLOAT,
            ast_pct_3p FLOAT,
            ast_pct_ft FLOAT,
            ast_ratio FLOAT,
```

```
            ast_to_ratio FLOAT,
            or_pct FLOAT,
            or_pct_after_2p FLOAT,
            or_pct_after_3p FLOAT,
            or_pct_after_ft FLOAT,
            dr_pct FLOAT,
            dr_pct_after_2p FLOAT,
            dr_pct_after_3p FLOAT,
            dr_pct_after_ft FLOAT,
            tr_pct FLOAT,
            st_pct FLOAT,
            blk_pct FLOAT,
            blk_pct_2p FLOAT,
            blk_pct_3p FLOAT,
            pf_100_poss FLOAT,
            df_100_poss FLOAT,
            per FLOAT,
            off_win_share FLOAT,
            def_win_share FLOAT,
            win_share FLOAT,
            win_share_per_40 FLOAT,
            obpm FLOAT,
            dbpm FLOAT,
            bpm FLOAT,
            vorp FLOAT,
            tm_pace_on FLOAT,
            tm_off_rtg_on FLOAT,
            tm_def_rtg_on FLOAT,
            tm_net_rtg_on FLOAT,
            tm_ts_pct_on FLOAT,
            tm_or_pct_on FLOAT,
            tm_to_pct_on FLOAT,
            tm_ft_ratio_on FLOAT,
            opp_ts_pct_on FLOAT,
            opp_or_pct_on FLOAT,
            opp_to_pct_on FLOAT,
            opp_ft_ratio_on FLOAT,
            tm_pace_off FLOAT,
            tm_off_rtg_off FLOAT,
            tm_def_rtg_off FLOAT,
            tm_net_rtg_off FLOAT,
            tm_ts_pct_off FLOAT,
            tm_or_pct_off FLOAT,
            tm_to_pct_off FLOAT,
            tm_ft_ratio_off FLOAT,
            opp_ts_pct_off FLOAT,
            opp_or_pct_off FLOAT,
            opp_to_pct_off FLOAT,
            opp_ft_ratio_off FLOAT,
            tm_pace_net FLOAT,
            tm_off_rtg_net FLOAT,
            tm_def_rtg_net FLOAT,
            tm_net_rtg_net FLOAT,
            tm_ts_pct_net FLOAT,
            tm_or_pct_net FLOAT,
            tm_to_pct_net FLOAT,
            tm_ft_ratio_net FLOAT,
            opp_ts_pct_net FLOAT,
            opp_or_pct_net FLOAT,
            opp_to_pct_net FLOAT,
            opp_ft_ratio_net FLOAT
        )
        """
    ]

    for command in table_commands:
        cur.execute(command)
    print("Tablas creadas en la base de datos avanzadas.")

    cur.close()
```

```
        conn.close()

    except psycopg2.Error as e:
        print(f"Error de base de datos")
        if conn:
            conn.close()

def clean_and_load_data():
    conn = psycopg2.connect(**ADVANCED_DB_CONFIG)
    conn.autocommit = True
    cur = conn.cursor()

    seasons_to_insert = ["2023-24", "2024-25"]
    for season in seasons_to_insert:
        cur.execute(
            """
            INSERT INTO seasons (season_id, season_name)
            VALUES (%s, %s)
            ON CONFLICT (season_id) DO NOTHING;
            """,
            (season, season)
        )

    for key, file_path in CSV_FILES.items():
        if os.path.exists(file_path):
            df = pd.read_csv(file_path, delimiter=';')
            season_year = '20' + key.split('_')[0] + '-' + key.split('_')[1]

            # Limpiar datos numéricos y reemplazar "-" por NaN
            for col in df.select_dtypes(include='object').columns.difference(['tm_name', 'name',
'role', 'nat']):
                df[col] = df[col].replace("-", None)
                df[col] = df[col].astype(str).str.replace(',', '.', regex=False)
                df[col] = pd.to_numeric(df[col], errors='coerce')

            # Diferenciar: stats -> NaN=0, atributos -> mantener NaN
            stats_cols = [c for c in df.columns if c not in ['tm_name', 'name', 'role', 'nat',
'height', 'age']]
            df[stats_cols] = df[stats_cols].fillna(0)

            if 'teams' in key:
                # Insertar equipos
                for _, row in df.iterrows():
                    cur.execute(
                        """
                        INSERT INTO teams (tm_name, season_id)
                        VALUES (%s, %s)
                        ON CONFLICT (tm_name) DO NOTHING;
                        """,
                        (row['tm_name'], season_year)
                    )

                # Insertar estadísticas de equipos
                for _, row in df.iterrows():
                    stats_columns = [col for col in df.columns if col not in ['tm_name']]
                    stats_values = [row['tm_name'], season_year] + [row[col] for col in
stats_columns]
                    placeholders = ', '.join(['%s'] * len(stats_values))

                    cur.execute(
                        f"""
                        INSERT INTO team_stats (tm_name, season_id, {', '.join(stats_columns)})
                        VALUES ({placeholders});
                        """,
                        stats_values
                    )

            elif 'players' in key:
                # Filtrar jugadores sin impacto
                if 'gp' in df.columns and 'min' in df.columns:
```

```
                    df = df[(df['gp'] > 3) & (df['min'] > 10)]

                # Insertar jugadores
                for _, row in df.iterrows():
                    cur.execute(
                        """
                        INSERT INTO players (name, role, nat, height, age, tm_name, season_id)
                        VALUES (%s, %s, %s, %s, %s, %s, %s)
                        ON CONFLICT (name) DO NOTHING;
                        """,
                        (row['name'], row['role'], row['nat'], row['height'], row['age'],
row['tm_name'], season_year)
                    )

                # Insertar estadísticas de jugadores
                for _, row in df.iterrows():
                    stats_columns = [col for col in df.columns if col not in ['name', 'role',
'nat', 'height', 'age', 'tm_name']]
                    stats_values = [row['name'], row['tm_name'], season_year, row['role'],
row['nat'], row['height'], row['age']] + [row[col] for col in stats_columns]
                    placeholders = ', '.join(['%s'] * len(stats_values))

                    cur.execute(
                        f"""
                        INSERT INTO player_stats (name, tm_name, season_id, role, nat, height,
age, {', '.join(stats_columns)})
                        VALUES ({placeholders});
                        """,
                        stats_values
                    )

        else:
            print(f"Advertencia: El archivo {file_path} no fue encontrado.")

    cur.close()
    conn.close()
    print("Datos cargados exitosamente.")

if __name__ == '__main__':
    create_advanced_database_and_tables()
    clean_and_load_data()
```

2.3. Capa Backend

La capa de Backend, construida con FastAPI como una API REST, es el corazón lógico de la aplicación. Es la responsable de procesar todas las solicitudes del frontend, interactuar con la base de datos y ejecutar la lógica de negocio compleja, incluyendo el chatbot inteligente y el modelo predictivo.

2.3.1. Estructura de endpoints

El backend se organiza en seis grupos de endpoints principales, cada uno dedicado a una funcionalidad específica de la plataforma.

2.3.1.1. Chat

Este endpoint (`/chat`) es la puerta de entrada a la funcionalidad de chatbot. Permite al usuario interactuar con el sistema en lenguaje natural. Recibe la pregunta del usuario y, utilizando un sistema multi-agente, determina la intención de la consulta (pregunta sobre datos, explicación de términos o análisis de partido por URL) y genera una respuesta coherente y detallada. Además, mantiene un contexto de conversación para ofrecer una experiencia más fluida.

2.3.1.2. Player Report

El endpoint de informe de jugador (`/player-report`) facilita la recuperación de estadísticas detalladas para uno o varios jugadores. El usuario puede solicitar las estadísticas de un jugador específico en una temporada determinada, o incluso comparar las estadísticas de dos jugadores. También ofrece la posibilidad de obtener las estadísticas promedio de todos los jugadores en una temporada concreta, lo que es útil para contextualizar el rendimiento individual.

2.3.1.3. Team Report

Similar al Player Report, el endpoint de informe de equipo (`/team-report`) permite obtener estadísticas completas para uno o dos equipos en temporadas específicas. Esto es ideal para analizar el rendimiento colectivo o realizar comparativas directas entre equipos, incluyendo también la opción de obtener promedios de la liga para una temporada.

2.3.1.4. Search Similar

El endpoint de búsqueda de jugadores similares (`/search-similar`) es una herramienta de scouting muy potente. Dado un jugador y una temporada, el sistema busca otros jugadores en la base de datos que tengan un perfil estadístico similar. Ofrece opciones para filtrar por el mismo rol y permite al usuario ponderar la importancia de diferentes métricas, lo que afina la búsqueda según sus criterios.

2.3.1.5. Performance Prediction

El endpoint de predicción de rendimiento (`/performance-prediction`) es donde reside la inteligencia predictiva de la aplicación. Permite al usuario seleccionar un quinteto de cinco jugadores (especificando jugador y temporada para cada uno) y, a partir de sus estadísticas

individuales, el modelo de Machine Learning entrenado predice el Net Rating que ese quinteto podría alcanzar si jugaran juntos.

2.3.1.6. Database

Este grupo de endpoints (`/database`) proporciona acceso a datos crudos y metadatos de la base de datos. Incluye funcionalidades para obtener listas de todos los jugadores, equipos y temporadas disponibles, así como los nombres de todas las estadísticas de jugadores con las que se está trabajando. Esto es esencial para poblar los desplegables y selectores en el frontend, asegurando que el usuario siempre elija opciones válidas.

2.4. Capa Frontend

La capa de Frontend, desarrollada con Streamlit, es la cara visible de la aplicación. Su objetivo es proporcionar una interfaz de usuario intuitiva y atractiva que permita a los entrenadores y analistas interactuar fácilmente con todas las funcionalidades del backend, transformando datos complejos en información comprensible y accionable

2.4.1. Estructura de páginas

El frontend se organiza en una página principal y varias páginas dedicadas a cada una de las funcionalidades clave, facilitando al usuario una navegación clara y estructurada.

2.4.1.1. Home

La página de inicio es el punto de entrada a la aplicación. Ofrece una bienvenida al usuario y una breve descripción de las capacidades de la plataforma. Es el centro de navegación desde donde el usuario puede acceder a todas las demás funcionalidades a través de un menú lateral o enlaces directos.

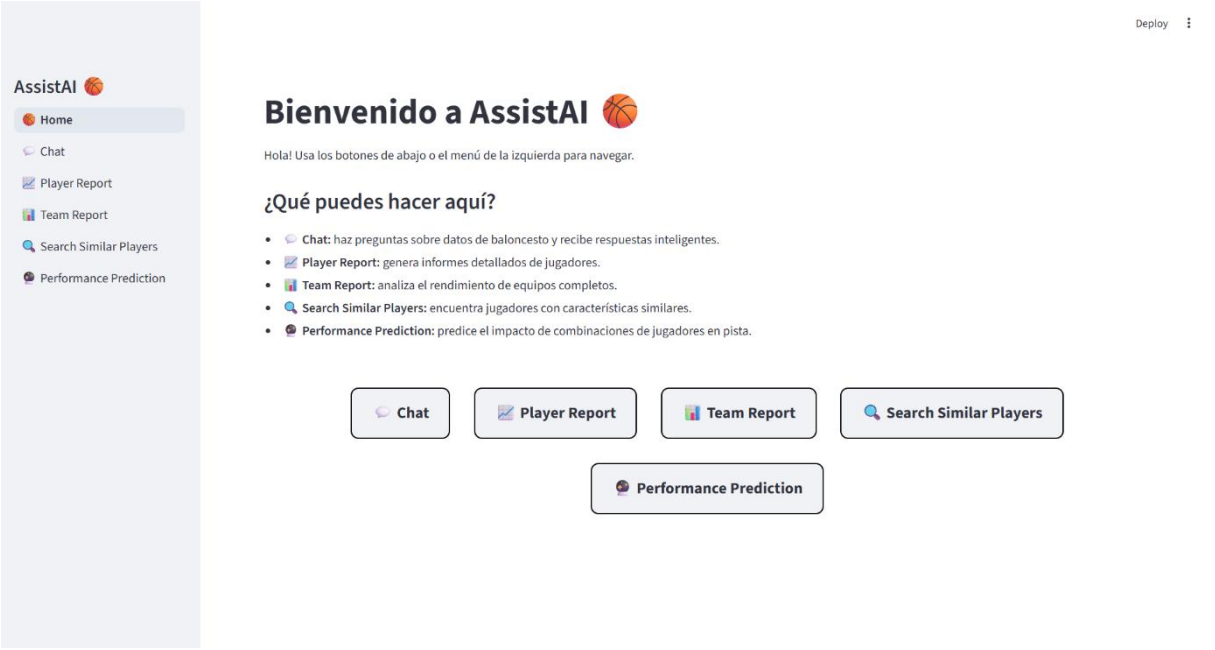

Figura 1. Página principal AssitAI

2.4.1.2. Chat

Esta página alberga la interfaz del chatbot. Presenta un primer mensaje de bienvenida y, tras esto, un área de texto donde el usuario puede escribir sus preguntas en lenguaje natural y un espacio para mostrar las respuestas generadas por el sistema. La interacción es similar a la de cualquier aplicación de mensajería, con la capacidad de mantener una conversación fluida gracias a la memoria contextual del chatbot.

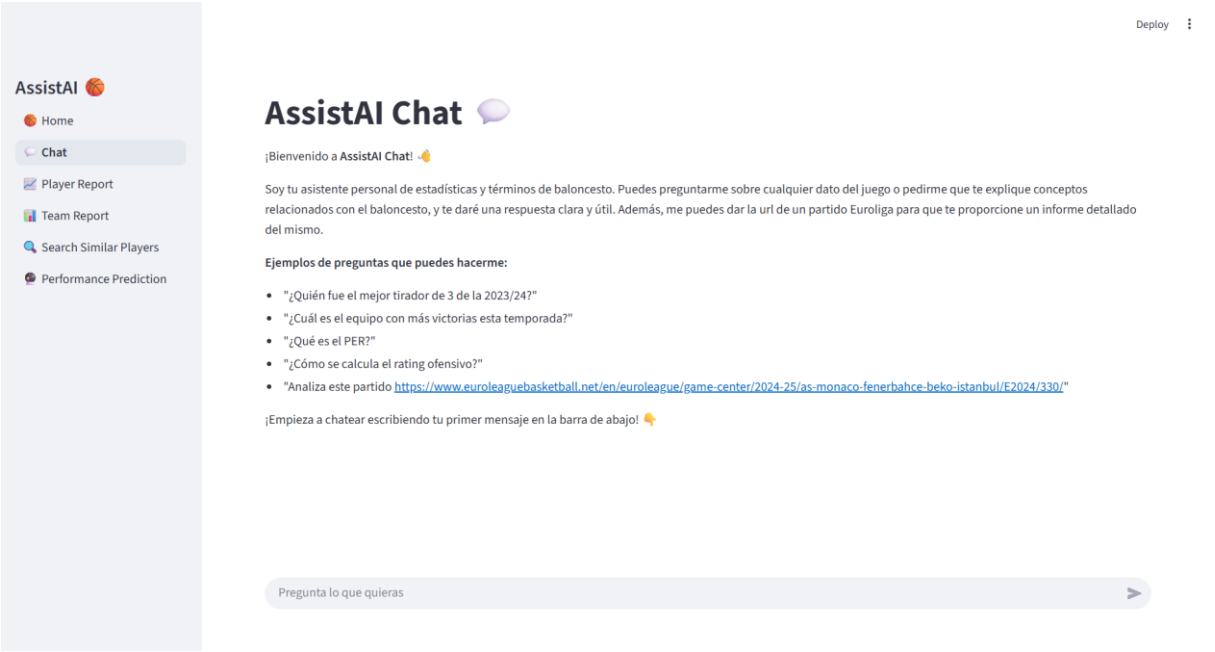

Figura 2. Página principal AssistAI - Chat

2.4.1.3. Player Report

En esta sección, los usuarios pueden generar informes detallados sobre jugadores individuales. Permite seleccionar uno o dos jugadores y sus respectivas temporadas para visualizar sus estadísticas clave, tanto básicas como avanzadas, a través de gráficos interactivos. Es ideal para un análisis profundo del rendimiento individual o para comparar directamente a dos jugadores.

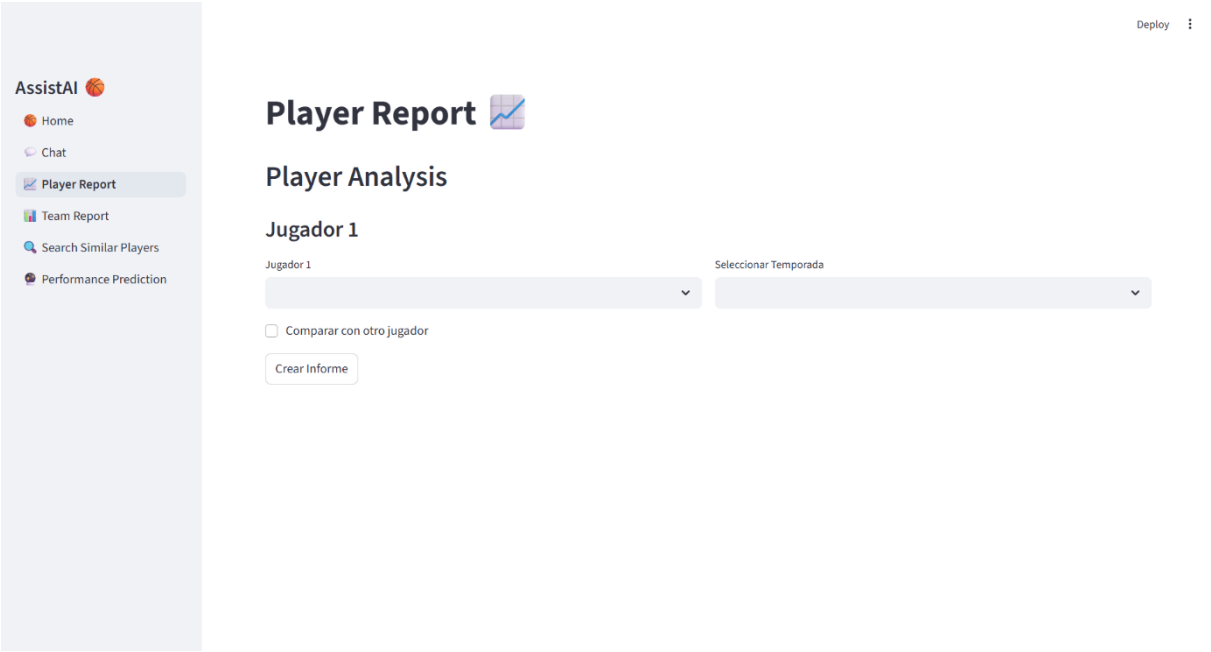

Figura 3. Página principal AssitAI - Player Report

2.4.1.4. Team Report

Similar al Player Report, esta página se enfoca en el análisis de equipos. Los usuarios pueden seleccionar uno o dos equipos y sus temporadas para obtener informes completos de sus estadísticas colectivas. Los dashboards interactivos permiten comparar el rendimiento de los equipos en diversas métricas, ofreciendo una visión global de su desempeño.

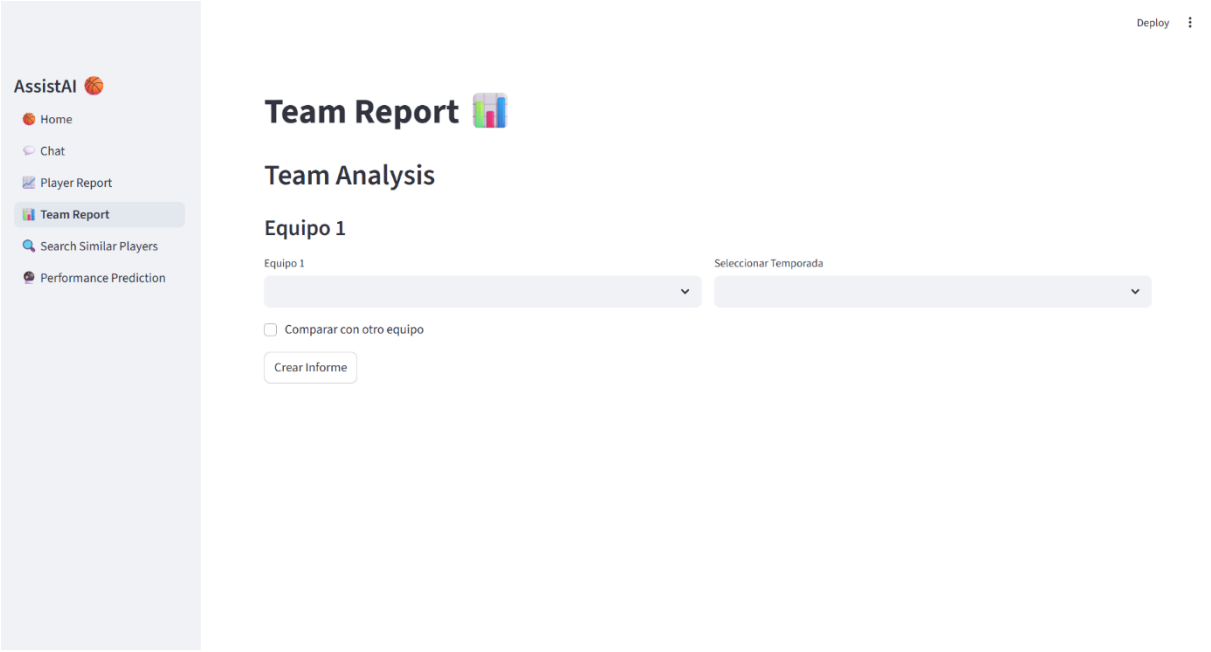

Figura 4. Página principal AssitAI - Team Report

2.4.1.5. Search Similar Players

Esta página proporciona la interfaz para la herramienta de búsqueda de jugadores similares. El usuario introduce el nombre de un jugador y una temporada, y la aplicación devuelve una lista de jugadores con perfiles estadísticos parecidos. Se ofrecen opciones para refinar la búsqueda, como filtrar por rol o ajustar la ponderación de diferentes estadísticas, lo que permite una personalización avanzada de los criterios de similitud.

Figura 5. Página principal AssitAI - Búsqueda Jugadores

2.4.1.6. Performance Prediction

La página de predicción de rendimiento permite a los usuarios experimentar con la formación de quintetos. Aquí, se pueden seleccionar cinco jugadores (con sus temporadas) y el sistema, utilizando el modelo de Machine Learning del backend, predice el Net Rating esperado para esa combinación. Es una herramienta valiosa para la planificación táctica y la simulación de escenarios.

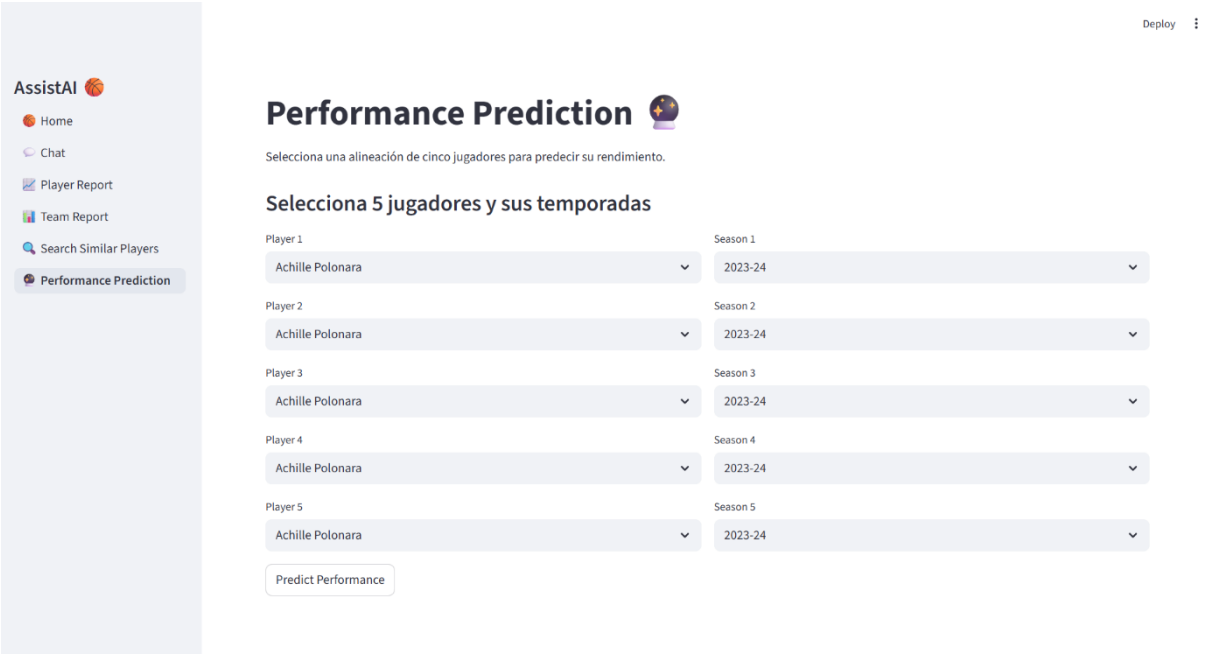

Figura 6. Página principal AssitAI - Predictor de Rendimiento

Además de estas páginas principales, en `src/frontend/utils` se encuentran módulos auxiliares como `dashboard_player` y `dashboard_team`. Estos ficheros son cruciales para la generación dinámica de los dashboards, creando gráficos complejos y comparativas visuales que enriquecen la experiencia del usuario. Estos dashboards no solo muestran datos, sino que también contextualizan el rendimiento del jugador/equipo comparándolo con las medias de la liga, ofreciendo una perspectiva más completa. La navegación en todas las páginas está diseñada para ser muy intuitiva, permitiendo un uso completo y sencillo de todas las capacidades del backend.

En un principio desarrollamos el inicio de sesión con usuario y contraseña a la aplicación. Sin embargo, finalmente, la idea fue desechada ya que nos daba problemas con los botones de la página principal y, además, no nos aportaba información extra que fuéramos a usar.

A su vez, gracias a la librería `st-theme` podemos detectar el tema que tiene el usuario para poder mostrar la aplicación acorde a la elección de este.

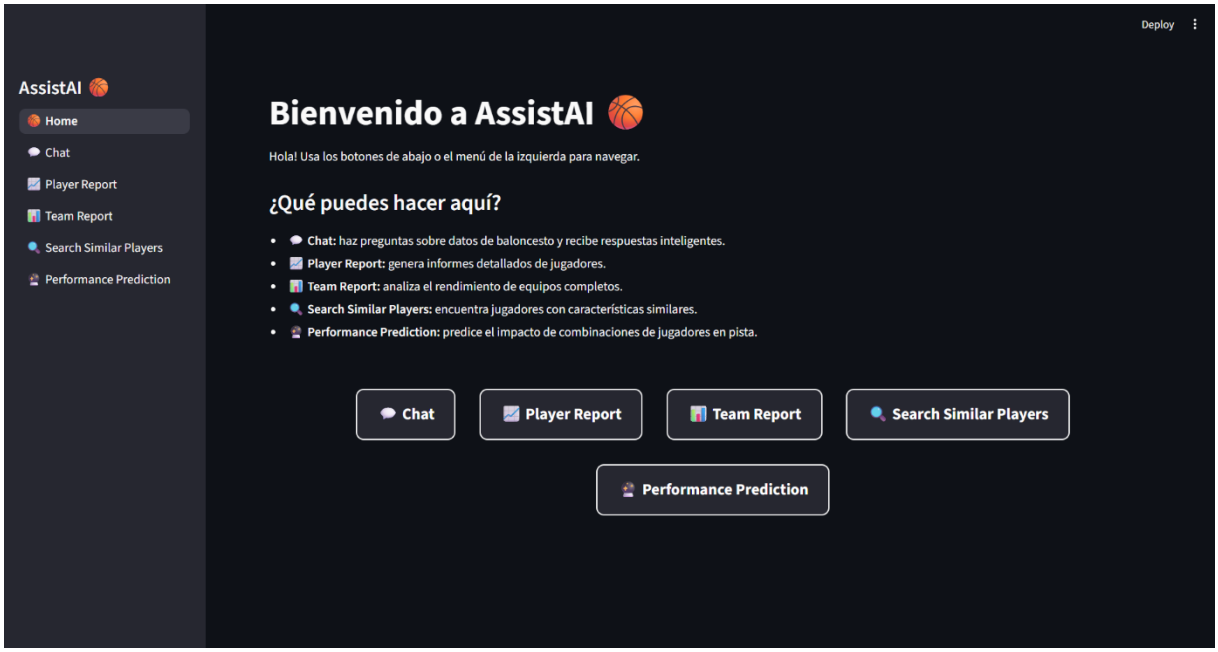

Figura 7. Página principal AssitAI (tema oscuro)

3. Funciones principales

Ahora, profundicemos en el detalle de cada una de las funcionalidades clave de la plataforma, explicando su implementación y el valor que aportan.

3.1. Chat

El chatbot es una de las funcionalidades más innovadoras y accesibles de la plataforma. Se ha implementado utilizando un **sistema multi-agente** basado en CrewAI, lo que le confiere una gran flexibilidad y capacidad para manejar diversas tipologías de consultas. Un sistema multi-agente implica la colaboración de diferentes "agentes" de IA, cada uno con un rol y un objetivo específico, que trabajan juntos para resolver una tarea compleja. En nuestro caso, esto permite al chatbot no solo responder preguntas directas, sino también interpretar la intención del usuario y derivar la consulta al agente más adecuado.

El flujo general del chat se gestiona en `src/backend/chat/crew.py` y `src/backend/chat/main.py`. El `main.py` es el encargado de definir los endpoints que tiene el chat, uno para crear el id de la conversación (usado para que la memoria no se comparta con otros chats) y el querecibe la pregunta del usuario y el contexto de la conversación (gestionado por la clase `Memory` en `src/backend/chat/memory.py`). Luego, un modelo de lenguaje (LLM) determina la rama a seguir (`general`, `stat_explainer` o `boxscore`) basándose en la pregunta y el contexto. Este mismo modelo es el encargado de hacer funcionar cada uno de los agentes, así como la crew al completo. En función de la rama a seguir la crew se modifica, ya que cada respuesta necesita de diferentes agentes para ser respondida de manera correcta.

Cabe destacar el funcionamiento de CrewAI. Como mencionamos anteriormente, se trata de un framework centrado en el desarrollo de sistemas multi-agente. Para ello, se ha de configurar mediante un prompt tanto cada agente como cada task que queramos asignarle a ese agente, esto lo hace por medio de ficheros `yaml`. Una vez definidos todos los integrantes del equipo de trabajo (o crew) así como las tareas a desarrollar por cada uno de ellos, podemos pasar a crear la crew, en la que tendremos que indicar qué orden se ha de seguir, secuencial (orden marcado por el desarrollador) o jerárquico (se usa un agente que determina el orden para cada petición) y qué LLM se usará. Así, el sistema hará uso de los prompts definidos en los ficheros `yaml` para dar solución a las peticiones que reciba.

3.1.1. Preguntas con datos

Cuando el usuario formula una pregunta que requiere la consulta de la base de datos (por ejemplo, "¿Quién fue el mejor tirador de 3 de la 2023/24?"), el sistema activa una colaboración entre dos agentes principales:

- **Agente `sql_dev`**: Este agente (`sql_dev` en `config/agents.yaml`) tiene el rol de "Senior Database Developer". Su objetivo es construir y ejecutar una consulta SQL (`sql_query`) basada en la solicitud del usuario. Es crucial que la consulta sea simple, eficiente y bien formada. Para ello, tiene acceso a la herramienta `NL2SQLTool` (definida en `src/backend/chat/crew.py`), que le permite traducir lenguaje natural a SQL. El `sql_dev` conoce el esquema de la base de datos (tablas `seasons`, `teams`, `players`, `team_stats`, `player_stats` y sus columnas, como se detalla en su `backstory` en `config/agents.yaml`), lo que le permite generar consultas precisas. Una vez que la consulta se ejecuta a través de la `NL2SQLTool`, el `sql_dev` proporciona un resumen de los resultados para que el siguiente agente sepa qué hacer con ellos.
- **Agente `writer`**: Una vez que el `sql_dev` ha recuperado los datos, el agente `writer` (definido en `config/agents.yaml` como "Basketball Report Writer and Chat Assistant") toma el relevo. Su objetivo es responder a la pregunta del usuario y escribir informes detallados basados en las estadísticas proporcionadas, siempre en español. El `writer` recibe el contexto de la conversación, la pregunta original del usuario y los datos recuperados de la base de datos. Su `backstory` lo capacita para explicar estadísticas complejas de manera clara y estructurada, utilizando formatos como listas o tablas para mejorar la legibilidad.

Este flujo de trabajo secuencial (`sql_query -> writer_sql`) asegura que las preguntas basadas en datos se respondan con precisión y de manera comprensible.

3.1.2. Explicación de términos estadísticos

Si la pregunta del usuario es una solicitud de explicación de un término estadístico de baloncesto (por ejemplo, "¿Qué es el Net Rating?"), el sistema activa el agente `stats_explainer`.

- **Agente `stats_explainer`**: Este agente (definido en `config/agents.yaml` como "Basketball Statistics Explainer") tiene como objetivo explicar conceptos y estadísticas

de baloncesto de manera fácil de entender. Su `backstory` incluye un glosario exhaustivo de términos comunes de baloncesto (GP, MIN, ORTG, DRTG, NRTG, USG%, eFG%, TS%, PER, WS, BPM, VORP, etc.), lo que le permite proporcionar definiciones claras, concisas e informativas. El agente utiliza este conocimiento para desglosar estadísticas complejas en términos simples, ayudando a los usuarios a comprender los matices del juego.

Esta rama del chat es directa y se enfoca en la educación del usuario sobre la terminología del baloncesto.

3.1.3. Informe de un partido

Una funcionalidad avanzada del chatbot es la capacidad de generar un análisis detallado de un partido de la Euroliga a partir de una URL de la página oficial de la Euroliga de ese partido proporcionada por el usuario.

- **Agente `boxscore_extractor`**: Este agente (definido en `config/agents.yaml` como "Boxscore Extractor") es el primero en actuar en esta rama. Su objetivo es extraer las estadísticas del boxscore de la URL del partido. Para ello, utiliza una herramienta personalizada llamada `BoxScoreTool` (definida en `src/backend/chat/tools/boxscore.py`). Esta herramienta es capaz de parsear la URL para obtener el `season` y `gamecode`, y luego utiliza la librería `euroleague_api` para obtener los datos crudos del boxscore. Además, la `BoxScoreTool` tiene la lógica para calcular estadísticas avanzadas (como `POSS`, `PPP`, `ORTG`, `DRTG`, `NetRTG`, `eFG%`, `TS%`, etc.) para cada jugador y para los totales del equipo, asegurando que el `report_generator` tenga toda la información necesaria.
- **Agente `report_generator`**: Una vez que el `boxscore_extractor` ha proporcionado todos los datos del partido (básicos y avanzados), el agente `report_generator` (definido en `config/agents.yaml` como "Basketball Game Report Analyst") entra en acción. Su objetivo es crear un informe completo, analítico y profesional del partido. Este informe incluye:
  - **Comparativa de Equipos**: Análisis de métricas clave para ambos equipos.
  - **Análisis de Jugadores Influyentes**: Identificación y explicación de por qué ciertos jugadores fueron decisivos.

- **Influencia de las Métricas en el Resultado Final**: Cómo las estadísticas y el rendimiento afectaron el desenlace del juego.
- **Claves para la Victoria**: Tres puntos clave para el equipo ganador y recomendaciones para el equipo perdedor de cara a futuros partidos.
- **Mejor Jugador de Cada Equipo**: Justificación del jugador más destacado de cada bando.

Este proceso (`boxscore_extraction -> report_generation en config/tasks.yaml`) permite al usuario obtener un análisis de scouting detallado con solo proporcionar una URL.

La memoria contextual del chatbot, implementada en `src/backend/chat/memory.py`, permite que el chatbot recuerde las interacciones previas, lo que es crucial para mantener conversaciones fluidas y responder a preguntas de seguimiento, ya que el chat, tras contestar a la pregunta del usuario (independientemente del tipo que sea), propondrá cinco preguntas que le pueden interesar, ya sea para ampliar la respuesta dada o para continuar con la conversación. Así, por ejemplo, si el usuario pregunta "¿Quién fue el mejor tirador de 3?" y luego simplemente dice "1", el chatbot entiende que se refiere a la primera pregunta sugerida en la respuesta anterior.

3.1.4. Fragmentos de código

`src/backend/chat/crew.py`. Lógica principal del sistema multi-agéntico usando CrewAI

```
import os
import datetime

from crewai import Agent, Crew, Process, Task, LLM
from crewai.project import CrewBase, agent, crew, task
from crewai_tools import NL2SQLTool
from config import ADVANCED_DB_CONFIG, LLM_MODEL

from src.backend.chat.tools.boxscore import BoxScoreTool

llm = LLM(
    model=LLM_MODEL
)

@CrewBase
class AssistAI():
    """AssistAI crew"""

    db_uri =
f"postgresql://{ADVANCED_DB_CONFIG['user']}:{ADVANCED_DB_CONFIG['password']}@{ADVANCED_DB_CONFIG[
'host']}/{ADVANCED_DB_CONFIG['dbname']}"
    nl2sql = NL2SQLTool(db_uri=db_uri, result_as_answer=True)
    boxscore_tool = BoxScoreTool(result_as_answer=True)
```

```
    agents_config = 'config/agents.yaml'
    tasks_config = 'config/tasks.yaml'

    # BRANCH 1: SQL + Natural Language
    @agent
    def sql_dev(self) -> Agent:
        return Agent(
            config=self.agents_config['sql_dev'],
            verbose=True,
            tools=[self.nl2sql],
            llm=llm
        )

    @agent
    def writer(self) -> Agent:
        return Agent(
            config=self.agents_config['writer'],
            verbose=True,
            llm=llm
        )

    # BRANCH 2: Explicación de términos estadísticos
    @agent
    def stats_explainer(self) -> Agent:
        return Agent(
            config=self.agents_config['stats_explainer'],
            verbose=True,
            llm=llm
        )

    # BRANCH 3: Generación de informe de un partido
    @agent
    def boxscore_extractor(self) -> Agent:
        return Agent(
            config=self.agents_config['boxscore_extractor'],
            verbose=True,
            tools=[self.boxscore_tool],
            llm=llm
        )

    @agent
    def report_generator(self) -> Agent:
        return Agent(
            config=self.agents_config['report_generator'],
            verbose=True,
            llm=llm
        )

    # BRANCH 1: SQL + Natural Language
    @task
    def sql_query(self) -> Task:
        return Task(
            config=self.tasks_config['sql_query'],
        )

    @task
    def writer_sql(self) -> Task:
        return Task(
            config=self.tasks_config['writer_sql'],
        )

    # BRANCH 2: Explicación de términos estadísticos
    @task
    def stats_explanation(self) -> Task:
        return Task(
            config=self.tasks_config['stats_explanation'],
        )

    # BRANCH 3: Generación de informe de un partido
    @task
```

```
    def boxscore_extraction(self) -> Task:
        return Task(
            config=self.tasks_config['boxscore_extraction'],
        )

    @task
    def report_generation(self) -> Task:
        return Task(
            config=self.tasks_config['report_generation'],
        )

    @crew
    def crew_sql(self, id: str) -> Crew:

        self.id = id

        return Crew(
            agents=[
                self.sql_dev(),
                self.writer()
            ],
            tasks=[
                self.sql_query(),
                self.writer_sql()
            ],
            process=Process.sequential,
            verbose=True
        )

    @crew
    def crew_stats(self, id: str) -> Crew:
        self.id = id

        return Crew(
            agents=[
                self.stats_explainer()
            ],
            tasks=[
                self.stats_explanation()
            ],
            process=Process.sequential,
            verbose=True
        )

    @crew
    def crew_boxscore(self, id: str) -> Crew:
        self.id = id

        return Crew(
            agents=[
                self.boxscore_extractor(),
                self.report_generator()
            ],
            tasks=[
                self.boxscore_extraction(),
                self.report_generation()
            ],
            process=Process.sequential,
            verbose=True
        )

    def create_id(self):
        """Creates a unique ID for the crew"""
        id = str(hash(datetime.datetime.now().timestamp()))
        if id not in os.listdir("memory"):
            os.mkdir(f"memory/{id}")
        else:
            i = 0
            while id in os.listdir("memory") and i < 5:
                i += 1
```

```
                id = str(hash(datetime.datetime.now().timestamp()))
                if id not in os.listdir("memory"):
                    os.mkdir(f"memory/{id}")
                    break
            if i == 5:
                raise Exception("Could not create a unique ID for the crew")
        return id

    def get_branch(self, question: str, context: str) -> str:
        messages = [
            {"role": "system", "content": "You are an expert on conversation flows. Your task is
to determine which branch to follow based on the user's question."},
            {"role": "user", "content": f"""
            You are given a question from a user and the context of the conversation (this is the
last messages of the conversation) and the user question. You have to take into account that the
user question is related to the last messages of the conversation (the last parts of the
conversation context), if the user question is just a number, he is selecting an specific
question from the last message of the conversation, so take that into account.
            Determine the best branch to follow to answer the user's question.
            There are three branches available:

            1. General: The user is asking anything related to basketball stats, such as player
performance, team standings, or game results. For example: "¿Quién es el mejor tirador de 3?",
"¿Cuál es el equipo con más victorias esta temporada?"...
            2. Stat Explainer: The user is asking for an explanation of a specific basketball
statistic or term. For example: "¿Qué es el PER?", "¿Cómo se calcula el rating ofensivo?"...
            3. Boxscore: The user is asking for a boxscore report of a specific game, and gives
the url to that game. For example: "https://www.euroleaguebasketball.net/en/euroleague/game-
center/2023-24/real-madrid-panathinaikos-aktor-athens/E2023/333/", "Analiza este partido
https://www.euroleaguebasketball.net/en/euroleague/game-center/2023-24/real-madrid-panathinaikos-
aktor-athens/E2023/333/"

            Conversation context: {context}
            User question: {question}

            Return the intent as one of the following: 'general' or 'stat_explainer' or
'boxscore'. Respond with only one of these three options, without any additional text or
explanation.
            """}
        ]

        response = llm.call(messages=messages)

        return response.strip().lower()
```

src/backend/chat/tools/boxscore.py. Lógica de la herramienta usada para extraer los datos del boxscore de partidos Euroliga a partir de su URL.

```
import re
import pandas as pd
from euroleague_api.boxscore_data import BoxScoreData
from crewai.tools import BaseTool

COL_MAP = {
    "Points": "PTS",
    "FieldGoalsMade2": "2PTM",
    "FieldGoalsAttempted2": "2PTA",
    "FieldGoalsMade3": "3PTM",
    "FieldGoalsAttempted3": "3PTA",
    "FreeThrowsMade": "FTM",
    "FreeThrowsAttempted": "FTA",
    "OffensiveRebounds": "OR",
    "DefensiveRebounds": "DR",
```

```
    "TotalRebounds": "TR",
    "Assistances": "AST",
    "Steals": "ST",
    "Turnovers": "TO",
    "BlocksFavour": "BLK",
    "BlocksAgainst": "BLKA",
    "FoulsCommited": "PF",
    "FoulsReceived": "DF",
    "Valuation": "VAL",
    "Plusminus": "+/-",
    "Minutes": "MIN"
}

def parse_url(url: str):
    """
    Extrae season y gamecode desde la URL de Euroleague.
    Ejemplo:
    https://www.euroleaguebasketball.net/es/euroleague/game-center/2023-24/real-madrid-
panathinaikos-aktor-athens/E2023/333/
    -> season=2023, gamecode=333
    """
    match = re.search(r'/E(\d{4})/(\d{1,3})', url)
    if not match:
        raise ValueError(f"No se pudo parsear season y gamecode de la URL: {url}")
    season = int(match.group(1))
    gamecode = int(match.group(2))
    return season, gamecode

def compute_advanced_stats(row: pd.Series, opp_totals: pd.Series) -> pd.Series:
    """
    Calcula estadísticas avanzadas para una fila (jugador o totales).
    """
    try:
        # Extraer básicos
        pts = float(row.get("Points", row.get("PTS", 0)))
        fgm2 = float(row.get("FieldGoalsMade2", row.get("2PTM", 0))); fga2 =
float(row.get("FieldGoalsAttempted2", row.get("2PTA", 0)))
        fgm3 = float(row.get("FieldGoalsMade3", row.get("3PTM", 0))); fga3 =
float(row.get("FieldGoalsAttempted3", row.get("3PTA", 0)))
        ftm = float(row.get("FreeThrowsMade", row.get("FTM", 0))); fta =
float(row.get("FreeThrowsAttempted", row.get("FTA", 0)))
        orb = float(row.get("OffensiveRebounds", row.get("OR", 0))); drb =
float(row.get("DefensiveRebounds", row.get("DR", 0))); trb = float(row.get("TotalRebounds",
row.get("TR", 0)))
        ast = float(row.get("Assistances", row.get("AST", 0))); tov = float(row.get("Turnovers",
row.get("TO", 0))); stl = float(row.get("Steals", row.get("ST", 0)))
        blk = float(row.get("BlocksFavour", row.get("BLK", 0))); pf =
float(row.get("FoulsCommited", row.get("PF", 0)))

        # Del rival (totales)
        opp_pts = float(opp_totals.get("Points", opp_totals.get("PTS", 0)))
        opp_fga2 = float(opp_totals.get("FieldGoalsAttempted2", opp_totals.get("2PTA", 0)))
        opp_fga3 = float(opp_totals.get("FieldGoalsAttempted3", opp_totals.get("3PTA", 0)))
        opp_fta = float(opp_totals.get("FreeThrowsAttempted", opp_totals.get("FTA", 0)))
        opp_to = float(opp_totals.get("Turnovers", opp_totals.get("TO", 0)))
        opp_or = float(opp_totals.get("OffensiveRebounds", opp_totals.get("OR", 0))); opp_dr =
float(opp_totals.get("DefensiveRebounds", opp_totals.get("DR", 0))); opp_tr =
float(opp_totals.get("TotalRebounds", opp_totals.get("TR", 0)))
        opp_tpa = float(opp_totals.get("FieldGoalsAttempted3", opp_totals.get("3PTA", 0)));
opp_2pa = float(opp_totals.get("FieldGoalsAttempted2", opp_totals.get("2PTA", 0)))

        # Calcular fgm y fga
        fgm = fgm2 + fgm3
        fga = fga2 + fga3
        opp_fga = opp_fga2 + opp_fga3

        # Posesiones estimadas
        poss = fga + 0.44*fta - orb + tov
        opp_poss = opp_fga + 0.44*opp_fta - opp_or + opp_to
        poss = max(poss, 1)
```

```
        opp_poss = max(opp_poss, 1)

        # Advanced stats
        stats = {
            "2PT%": fgm2/fga2 if fga2 else 0,
            "3PT%": fgm3/fga3 if fga3 else 0,
            "FGM": fgm,
            "FGA": fga,
            "FG%": fgm/fga if fga else 0,
            "FT%": ftm/fta if fta else 0,
            "POSS": poss,
            "PPP": pts/poss,
            "ORTG": 100*pts/poss,
            "DRTG": 100*opp_pts/opp_poss if row["Player"] in ["TOTALS", "TEAM ADVANCED"] else
float('NaN'),
            "NetRTG": (100*pts/poss) - (100*opp_pts/opp_poss) if row["Player"] in ["TOTALS",
"TEAM ADVANCED"] else float('NaN'),
            "eFG%": (fgm + 0.5*fgm3)/fga if fga else 0,
            "TS%": pts / (2*(fga + 0.44*fta)) if (fga+0.44*fta) else 0,
            "FT Ratio": fta/fga if fga else 0,
            "TO%": tov/poss,
            "AST%": ast/fgm if fgm else 0,
            "AST/TO": ast/tov if tov else ast,
            "OR%": orb / (orb + opp_dr) if (orb+opp_dr) else 0,
            "DR%": drb / (drb + opp_or) if (drb+opp_or) else 0,
            "TR%": trb / (trb + opp_tr) if (trb+opp_tr) else 0,
            "ST%": stl/poss,
            "BLK%": blk / opp_2pa if opp_2pa else 0,
            "PF per 100 Poss": 100*pf/poss
        }
        return pd.Series(stats)
    except (TypeError, ValueError):
        # Manejar filas con datos no numéricos (e.g., 'DNP')
        return pd.Series({
            "POSS": float('NaN'),
            "PPP": float('NaN'),
            "ORTG": float('NaN'),
            "DRTG": float('NaN'),
            "NetRTG": float('NaN'),
            "eFG%": float('NaN'),
            "TS%": float('NaN'),
            "FT Ratio": float('NaN'),
            "TO%": float('NaN'),
            "AST%": float('NaN'),
            "AST/TO": float('NaN'),
            "OR%": float('NaN'),
            "DR%": float('NaN'),
            "TR%": float('NaN'),
            "ST%": float('NaN'),
            "BLK%": float('NaN'),
            "PF per 100 Poss": float('NaN')
        })

def get_boxscore_from_url(url: str):
    """
    A partir de una URL de Game Center, devuelve un DataFrame completo con
    estadísticas individuales y las estadísticas avanzadas del equipo.
    """
    season, gamecode = parse_url(url)
    raw = BoxScoreData().get_boxscore_data(season, gamecode, "Stats")

    if not isinstance(raw, list) or len(raw) < 2:
        raise ValueError("El boxscore no tiene el formato esperado (lista con dos equipos).")

    dfs = {}
    for item in raw:
        team_name = item.get("Team", "UNKNOWN").strip().upper()
        players = pd.DataFrame(item.get("PlayersStats", []))
        totals = pd.DataFrame([item.get("totr", {})])
```

```
        # Concatenar jugadores + fila de totales para un único DataFrame
        df_team = pd.concat([players, totals.assign(Player="TOTALS")], ignore_index=True)
        dfs[team_name] = df_team

    teams = list(dfs.keys())

    # Calcular totales del oponente
    opp_totals = {teams[i]: dfs[teams[1-i]][dfs[teams[1-i]]["Player"] == "TOTALS"].iloc[0] for i
in range(2)}

    # Aplicar estadísticas avanzadas
    result_dfs = []
    for team_name, df_team in dfs.items():
        opp_total_row = opp_totals[team_name]

        # Calcular y añadir stats avanzadas a cada fila
        advanced_stats_df = df_team.apply(lambda row: compute_advanced_stats(row, opp_total_row),
axis=1)
        df_team = pd.concat([df_team, advanced_stats_df], axis=1)

        # Fusión de filas 'TOTALS' y 'TEAM ADVANCED'
        totals_row = df_team[df_team["Player"] == "TOTALS"].index[0]
        # Aquí no hay 'TEAM ADVANCED' ya que se ha fusionado en TOTALS

        # Eliminar las filas de jugadores con 'DNP' si es necesario
        df_team = df_team[df_team["Minutes"] != "DNP"].reset_index(drop=True)

        df_team.insert(0, "TEAM TOTAL", team_name)
        result_dfs.append(df_team)

    final_df = pd.concat(result_dfs, ignore_index=True)

    # Renombrar columnas con las siglas
    final_df = final_df.rename(columns=COL_MAP)

    # Renombrar columnas
    final_df = final_df.rename(columns={
        "TEAM TOTAL": "Team",
        "Team": "TeamCode",
        "Player_ID": "PlayerID"
    })

    # Convertir IsStarter a bool
    if "IsStarter" in final_df.columns:
        final_df["IsStarter"] = final_df["IsStarter"].astype(bool)

    # Eliminar columnas no deseadas
    final_df = final_df.drop(columns=["IsPlaying"], errors="ignore")

    # Reordenar columnas para que empiecen con el orden deseado
    column_order = [
        'Team', 'TeamCode', 'Dorsal', 'Player', 'IsStarter', 'MIN', 'PTS', '2PTM', '2PTA',
'2PT%', '3PTM', '3PTA',
        '3PT%', 'FGM', 'FGA', 'FG%', 'FTM', 'FTA', 'FT%', 'OR', 'DR', 'TR', 'AST', 'TO', 'ST',
'BLK', 'BLKA',
        'PF', 'DF', 'VAL', '+/-', 'POSS', 'PPP', 'ORTG', 'DRTG', 'NetRTG', 'eFG%', 'TS%', 'FT
Ratio', 'TO%',
        'AST%', 'AST/TO', 'OR%', 'DR%', 'TR%', 'ST%', 'BLK%', 'PF per 100 Poss'
    ]

    # Reordenar las columnas del DataFrame
    final_df = final_df.reindex(columns=column_order)

    return final_df.to_json()

class BoxScoreTool(BaseTool):
    name: str = "BoxScoreTool"
    description: str = "Returns advanced team boxscore stats given a Euroleague URL"

    def _run(self, url: str):
```

```
        return get_boxscore_from_url(url)
```

A modo de ejemplo, dejo por aquí el prompt que recibe el agente encargado de extraer los datos del boxscore:

`src/chat/config/agents.yaml`:

```
boxscore_extractor:
  role: >
    Boxscore Extractor
  goal: >
    Extract boxscore statistics from basketball game URLs and compute advanced stats.
  backstory: >
    You are a specialized tool for extracting boxscore statistics from basketball game URLs. You
can parse the URL to obtain the season and game code, retrieve the boxscore data, and compute
advanced statistics for teams and players.
```

`src/chat/config/tasks.yaml`:

```
boxscore_extraction:
  description: >
    You are an assistant for basketball analysts. Your goal is to retrieve and present the
boxscore and advanced data for a specific game. The user will give you a URL to the game, with
that URL you will extract the boxscore information using the BoxScoreTool tool.
  expected_output: >
    All the data retreived from the boxscore of the game after using the tool BoxScoreTool.
  agent: boxscore_extractor
```

3.1.5. Capturas de pantalla

En esta sección veremos alguna captura de pantalla de un chat, mostrando de un modo visual la memoria contextual con la que cuenta esta función de la aplicación y cómo se desenvuelve ante preguntas usuales.

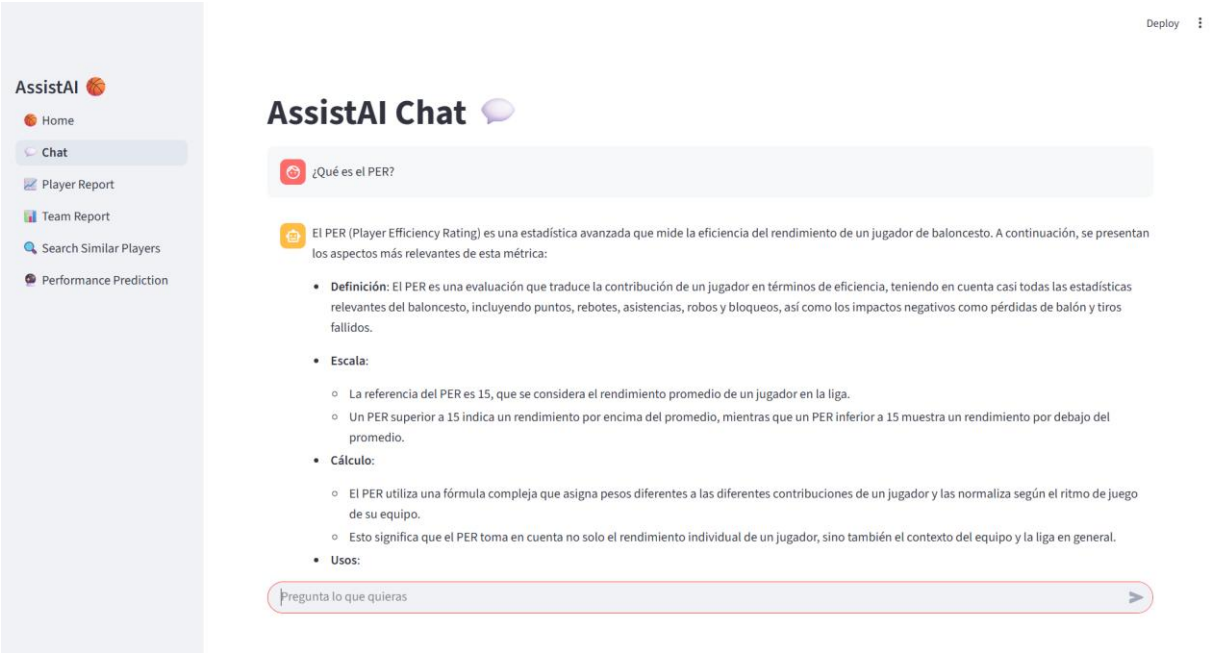

Figura 8. AssistAI - Chat - Explicación de términos

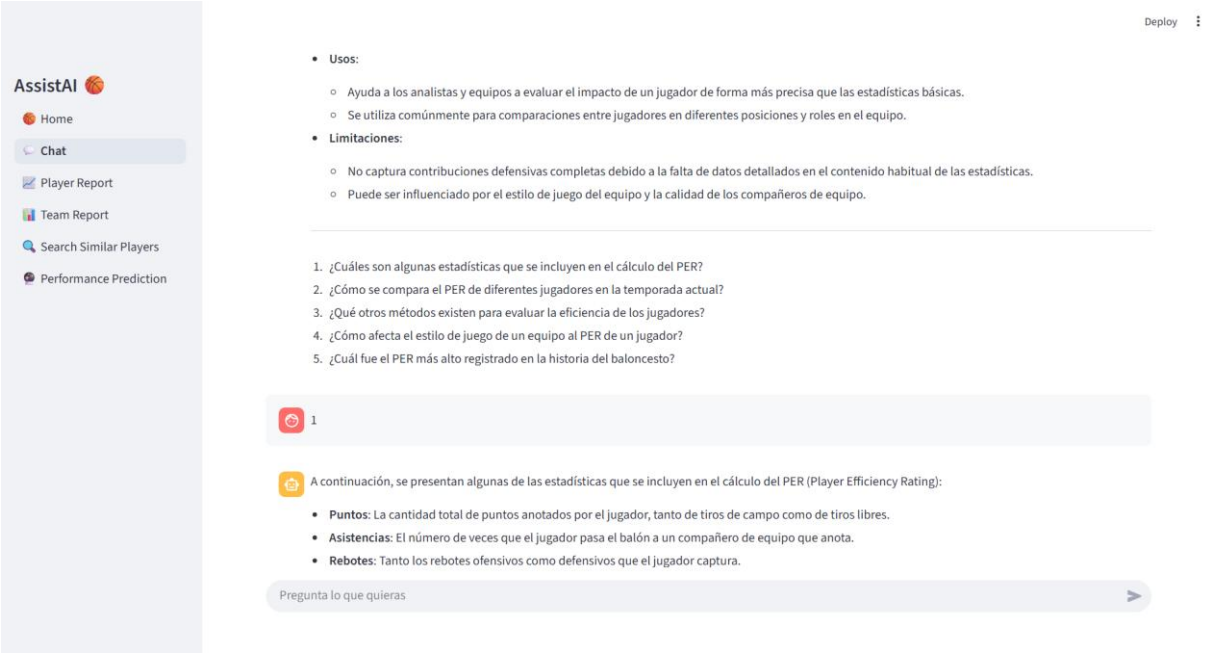

Figura 9. AssistAI - Chat - Uso de preguntas propuestas

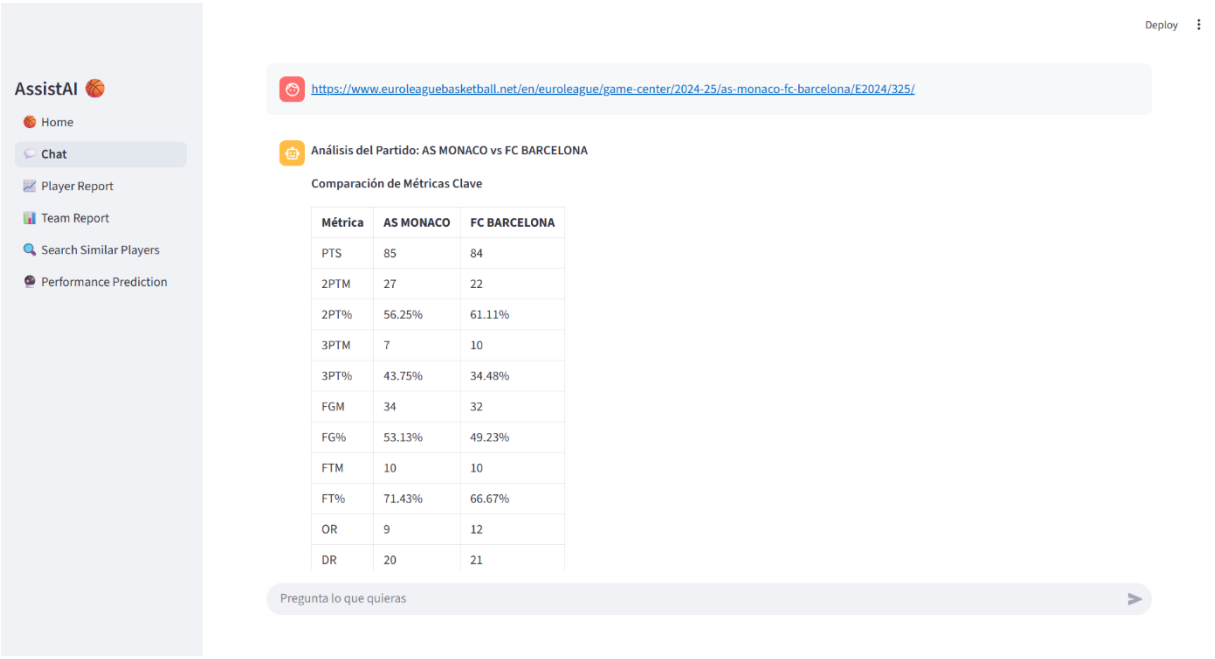

Figura 10. AssistAI - Chat - Análisis de un partido a partir de su URL

En la sección 4 se podrá apreciar desde le demo enlazada un flujo de conversación normal.

3.2. Player Report

La funcionalidad de Player Report, accesible a través del endpoint `/player-report` en el backend y su correspondiente página en el frontend, permite a los usuarios obtener y visualizar estadísticas detalladas de jugadores individuales. Es una herramienta esencial para el análisis de rendimiento, el scouting y la comparación de talentos.

3.2.1. Dashboard

El corazón de la funcionalidad de Player Report es el dashboard interactivo, diseñado para presentar una gran cantidad de información de manera clara y comprensible. Este dashboard se genera dinámicamente utilizando el módulo `dashboard_player` en `src/frontend/utils/dashboard_player.py`.

El usuario puede seleccionar uno o dos jugadores y sus respectivas temporadas. Si se selecciona un solo jugador, el dashboard mostrará sus estadísticas en comparación con el promedio de la liga para esa temporada, proporcionando un contexto valioso sobre su rendimiento relativo. Si se seleccionan dos jugadores, el dashboard ofrecerá una comparativa

directa entre ellos, además, si ambos jugadores están siendo comparados en la misma temporada, también se añadirá a la comparativa el promedio de la liga en esa temporada.

El dashboard cuenta con seis secciones, cada una de ellas basándose en una zona específica del juego, cada una de ellas, así como cada gráfico está diseñado para resaltar aspectos específicos del rendimiento del jugador. Estas secciones son:

1. **Perfil**. Se muestra una tarjeta con información básica del jugador (nombre, temporada, equipo, rol…) y algún dato general de su equipo en la temporada (partidos jugados, minutos por partido, récord y W%).
2. **Tiro**. Aquí usamos todos los datos que tenemos relacionados al lanzamiento a canasta del jugador, desde sus puntos por partido y porcentajes hasta la frecuencia por zonas de tiro, puntos por tiro y eFG% entre otras. Esta sección a su vez se divide en cinco secciones:
    1. **Eficiencia de tiro**. Aquí tenemos un gráfico de barras para comparar los puntos por partido de ambos jugadores (y la media de la liga, si procede). De igual manera, vemos un gráfico de radar para poder comparar sus porcentajes (2PT%, 3PT%, FT%, FG%, eFG% y TS%).
    2. **Volumen de tiro**. Nos encontramos con un gráfico de barras que compara los valores totales por partido de tiros intentados y anotados por categoría (de 2 puntos, triples, tiros de campo y tiros libres).
    3. **Frecuencia vs Eficiencia**. El gráfico más completo de esta sección nos muestra un gráfico de burbujas que compara la frecuencia de cada tipo de tiro según su distancia al aro (RIM, PAINT, MID, C3, L3) con los puntos por tiro que ese tipo tiro devuelve. Además, el tamaño nos indica los intentos totales por zona (estos intentos son aproximados ya que los calculamos con la frecuencia y el número de tiros totales).
    4. **Distribución de tiro**. Para comparar mejor aún la diferencia de tendencias entre los jugadores, vemos un gráfico de sectores con las frecuencias de tiro por zona de los jugadores.
    5. **Otros**. Finalmente, finalizamos el dashboard de tiro con un par de gráficos de barras, uno para los tapones recibidos (BLKA) y otro para el ratio de tiros libres (FT Rate).

3. **Uso y creación**. Esta parte ya no es tan directa como la del tiro, en esta sección nos encontramos principalmente las estadísticas que diríamos que pertenecen a la habilidad como *playmaker* del jugador, las secciones que nos encontramos son:
    1. **Uso y creación**. Dos gráficos de barras que comparan dos tipos de forma de crear juego. El primero es el formado por el porcentaje de uso (USG%), las posesiones (POSS) y los puntos por posesión (PPP), midiendo así la capacidad anotadora del jugador. Por otro lado, el segundo gráfico compara las asistencias, pérdidas y el ratio asistencia por pérdida (AST/TO), lo cual nos permite comparar la capacidad que tiene el jugador para crear juego para sus compañeros.
    2. **AST% vs TO%**. Este gráfico a pesar de ser muy sencillo me parece muy importante, ya que se trata de un gráfico de burbujas que compara el porcentaje de asistencias (AST%) con el porcentaje de pérdidas (TO%), usando los minutos para darle tamaño. Además, hemos añadido dos rectas auxiliares que marcan la media de la liga, partiendo el gráfico en cuatro cuadrantes y dejando más claro aún si el jugador está por encima o por debajo de la media.
    3. **Distribución de asistencias**. Vemos un gráfico de barras apiladas que compara los porcentajes de asistencia por tipo de tiro (2 puntos, triple o tiro libre), permitiendo ver cómo genera juego este jugador.
    4. **Control de pérdidas**. Volviendo al tema de las pérdidas, volvemos a tener dos gráficos de barras, ahora en este caso, uno que compara el TO% con el porcentaje de pérdidas con balón vivo (LTO%) y con balón muerto (DTO%). Mientras que el otro compara el ratio de asistencia (AST Ratio) con el ratio de asistencia por pérdida (AST/TO).
4. **Rebote**. Esta sección sea probablemente la más sencilla de todo el dashboard, pero no por ello es la menos importante ni mucho menos. Contamos únicamente con cuatro gráficos de barras divididas en dos secciones.
    1. **Rebote**. Dos gráficos que nos permiten comparar las estadísticas básicas del rebote, uno de ellos compara los valores totales (OR, DR y TR) mientras que el otro sus porcentajes (OR%, DR% y TR%).
    2. **Rebote por contexto de tiro**. Ahora, comparamos los porcentajes de rebote según el contexto del tiro anterior (tiro de 2, triple o tiro libre). Así, uno

comparará los porcentajes de rebotes ofensivos, mientras que el otro hará lo propio con los defensivos.

5. **Defensa**. Pasando a la próxima sección, nos encontramos varios gráficos de barras y alguna métrica un poco más avanzada que hemos de revisar.
    1. **Robos y tapones**. La primera sección es la más general, con dos gráficos de barras comparando robos y tapones, en uno de ellos los totales mientras que en el otro los porcentajes.
    2. **Detallando los tapones**. Tenemos acceso al porcentaje de tapones por contexto de tiro (si ha sido taponando un tiro de 2 o un triple), por lo que podemos montar un gráfico de barras para la comparativa.
    3. **Disciplina defensiva**. Aquí, nos centramos en las faltas, comparando PF 100 Poss y DF 100 Poss, las cuales se tratan de faltas cometidas y faltas recibidas por 100 posesiones respectivamente.
    4. **Impacto sobre el rival**. Volvemos a ver un gráfico de burbujas. En este caso, usará los minutos como tamaño igual que hacía el anterior que vimos. Sin embargo, este comparará el TS% (True Shooting) y el TO% del rival mientras el jugador esté en pista, por lo que queremos que el OPP TS% sea cuanto más bajo posible mientras que al contrario, buscamos elevar el porcentaje de pérdidas rival.
6. **Impacto & Avanzadas**. Finalmente, llegamos a la última sección, la cual hace uso de las que probablemente sean las métricas más avanzadas con las que contamos en la BD, las dividimos en las siguientes secciones.
    1. **Impacto & Avanzadas**. Comenzamos con una de las métricas más extendidas que veremos en esta sección, la cual es el *rating*, ya sea ofensivo, defensivo o net rating. En este caso, comparamos la versión individual de los tres tipos. En el gráfico de la derecha vemos una comparativa de los tres tipos de *Box Plus Minus* (BPM) siendo estos ofensivo (OBPM), defensivo (DBPM) y total (BPM) con *Value Over Replacement Player* (VORP) mediante un gráfico de barras.

       El BPM es la evolución clásica del +/- al que tanto estamos acostumbrados, ya que simplemente se trata de la normalización de esta a 100 posesiones. En el caso de la ofensiva, a la hora de normalizar, se calcula con unos coeficientes tales que únicamente toman en cuenta la ofensiva del jugador. Igual sucede

para la defensiva. Cabe destacar que la suma del OBPM y DBPM nos da como resultado el BPM.

Por otro lado, el VORP o *Value Over Replacement Player* nos permite comprender la contribución de un jugador comparándola con la del jugador de reemplazo. El VORP considera los minutos y partidos jugador y considera el valor -2 como el BPM del jugador de reemplazo. Se podría decir pues que el VORP es la estadística más fiable para comprender qué jugadores contribuyen significativamente a las victorias de su equipo.

2. **Ratings individuales**. Nos encontramos ahora un gráfico de dispersión que compara el rating individual ofensivo y defensivo de cada jugador. Además, volvemos a ver las rectas que marcan la media de la liga, creando cuatro cuadrantes que nos determinan perfectamente en qué zona se sitúa cada jugador. En este caso, en el eje x vemos el rating ofensivo, por lo que cuanto más a la derecha se encuentre el jugador mejor y, al contrario para con el eje y, ya que vemos el rating defensivo, por lo que cuanto más abajo mejor. Así pues, podemos determinar cómo el mejor cuadrante en el que ir a parar es el de la esquina inferior derecha.
3. **Win shares**. Esta métrica nos permite comprender qué jugadores de la plantilla contribuyen más a las victorias. El principio es el siguiente: la victoria de cada equipo se considera que es 1 Win Share. Dentro de ese 1, cada jugador habrá contribuido de forma diferente a la victoria. Esta estadística estima la contribución a la victoria en términos numéricos. Tenemos Win Share ofensiva y defensiva y, con su suma, obtenemos la total. Además, también contamos con una normalización, Win Share per 40 nos determina el Win Share de un jugador por 40 minutos (un partido).

   Así pues, nos encontramos con un gráfico de barras que compara estas cuatro métricas.
4. **Contexto de equipo (ON vs OFF)**. Esta es una de las secciones que personalmente más importante me parece, ya que compara el rendimiento del equipo cuando el jugador está en pista de cuando no lo está. Primero, vemos tres gráficos de líneas que cada uno de ellos compara el rating del equipo (ofensivo, defensivo y net) en estas situaciones. Así pues, para el primero,

cuanto mayor pendiente descendiente tenga la línea más importante es ese jugador para el equipo a nivel ofensivo y viceversa en el rating defensivo. En el net rating buscamos que la pendiente sea descendente al igual que en el rating ofensivo.

Tras esto, vemos dos gráficos de barras. El primero está hecho para ver con mayor claridad y detalle la comparativa de ratings que hemos visto con los gráficos de líneas, ya que compara el net rating del equipo con el jugador en pista, con el mismo cuando el jugador no está en pista y muestra por último la diferencia de los dos, el cual es la diferencia total de que el jugador esté o no en pista. De igual manera, en el segundo, vemos el TS% del equipo cuando el jugador está en pista, cuando no lo está y finalmente la diferencia de los dos.

5. **Resumen rápido**. Finalizamos el dashboard y el análisis/comparativa de jugadores con tres métricas ya muy bien conocidas que sirven para dar una visión global del jugador. Estas son el PER (*Player Efficiency Rating*), la VAL y más/menos +/-. Son comparadas en un gráfico de barras usual.

3.2.2. Capturas de pantalla

Veremos ahora algunos de los gráficos que construye la aplicación cuando queremos comparar dos jugadores. En este caso, estamos comparando a los dos jugadores que encabezaron la carrera por el MVP de la Euroliga 2024/25, Kendrick Nunn y Sasha Vezenkov.

Mo veremos muchos gráficos por no sobrecargar el documento, en la demo se pueden apreciar todos ellos.

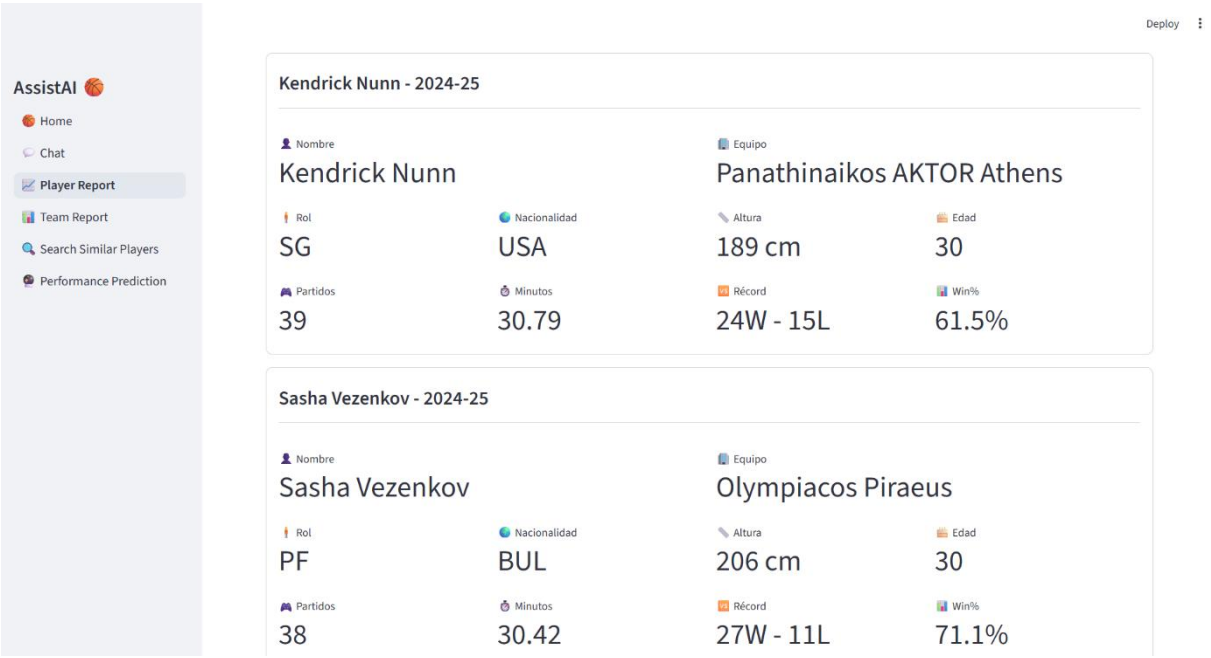

Figura 11. AssistAI - Player Report - Perfil de jugadores

Figura 12. AssistAI - Player Report - Eficiencia de tiro

Figura 13. AssistAI - Player Report - Impacto & Avanzadas

Figura 14. AssistAI - Player Report - Contexto de equipo

3.3. Team Report

La funcionalidad de Team Report, accesible a través del endpoint `/team-report` en el backend y su página dedicada en el frontend, ofrece una visión completa del rendimiento colectivo de los equipos. Es una herramienta indispensable para el análisis de tendencias de equipo, la evaluación de estrategias y la comparación con rivales.

3.3.1. Dashboard

Al igual que con los jugadores, el Team Report se centra en un dashboard interactivo y personalizable, generado por el módulo `dashboard_team` en `src/frontend/utils/dashboard_team.py`. Su funcionamiento es exactamente el mismo que el de Team Report, salvo que ahora con las estadísticas globales del equipo en vez de las individuales del jugador.

El usuario tiene la flexibilidad de seleccionar uno o dos equipos y sus respectivas temporadas. Si se elige un solo equipo, el dashboard presentará sus estadísticas en relación con el promedio de la liga para esa temporada, ofreciendo un benchmark claro de su desempeño. Si se seleccionan dos equipos, el dashboard facilitará una comparativa directa, destacando sus fortalezas y debilidades relativas, en caso de ser de la misma temporada, también añadirá las medias de esta temporada a la comparativa.

El dashboard cuenta con las mismas seis secciones que contaba el de jugadores y he intentado que sean lo más parecidos posibles para facilitar así la familarización con la aplicación y el uso más eficiente de esta. Así pues, la explicación en este caso será mucho más breve.

1. **Perfil**. Tarjeta básica del equipo (nombre, temporada, partidos jugados) y algún dato general como son los minutos de media, el récord y el porcentaje de victorias.
2. **Tiro**.
    1. **Eficiencia de tiro**. Comparativa de puntos y porcentajes de tiro.
    2. **Volumen de tiro**. Comparativa de tiros totales (intentados y anotados).
    3. **Frecuencia vs Eficiencia**. Frecuencia por zona vs PPT.
    4. **Distribución de tiro**. Frecuencia de tiro por zona del campo.
    5. **Oportunidades de tiro – Acciones**. Al tratarse de estadísticas de equipos globales, algo que hay que tener muy en cuenta son las oportunidades de tiro, también conocidas como acciones, ya que eso nos da una buena visión del

juego del equipo, dándonos las veces que pueden convertir esa acción en puntos. En este caso, damos el valor normalizado por 100 posesiones.

6. **Volumen de acciones vs eficiencia**. Conociendo esta métrica de SC normalizada, la podemos comparar con un porcentaje de tiro (eFG% en este caso) para tener una visión de cuánto aprovechan esas oportunidades. Lo hacemos mediante un gráfico de burbujas en el que el tamaño es el número de tiros de campo intentados.
7. **Otros**. BLKA y FT Ratio.

3. **Uso y creación**. Así como antes en esta sección veíamos cuánta habilidad como *playmaker* tenía el jugador, ahora se podría decir que esta sección nos indica cómo de rápido juega el equipo y cuánto comparten la bola.
    1. **Ritmo y creación**. Ahora, comparamos el Pace (ritmo) con las posesiones, aunque prácticamente son las mismas (ya que Pace es la métrica de posesiones normalizada a 40 minutos) nunca está de más tener ambos datos para un análisis más exhaustivo. El segundo gráfico es el mismo que veíamos antes de AST, TO y AST/TO.
    2. **Eficiencia por posesión**. El propio nombre nos indica todo, conocemos las posesiones del equipo, ahora bien, ¿qué tan bien (o mal) son capaces de aprovechar esas posesiones? Pues eso nos lo dicen los puntos por posesión (PPP) que anotan (OFF PPP) y que reciben (DEF PPP). Cuanto más alto sea la primera y más baja la segunda, mejor.
    3. **Relación Pace Vs OFF PPP**. Este gráfico es algo que genera debate en todos los lados, porque, ¿es mejor jugar más rápido o más lento? Es un debate largo y tendido en el que este gráfico puede ayudar a entender algo mejor el juego de cada equipo, ya que compara el ritmo con los puntos por posesión anotados (ofensivos), así pues, a mayor ritmo y mayor OFF PPP tu equipo anotará al final del partido más puntos.
    4. **AST% vs TO%**.
    5. **Distribución de asistencias**. Porcentaje de asistencias con contexto del tiro.
    6. **Control de pérdidas**. Comparativa de porcentaje de pérdidas y comparativa del AST Ratio y AST/TO.

4. **Rebote**. Vuelve a estar esta sección en las mismas condiciones que en el Team Report, siendo la más sencilla, es exactamente igual que la anterior, comparándolas estadísticas básicas y por contexto de tiro.
    1. **Rebote**.
    2. **Rebote por contexto de tiro**.
5. **Defensa**. En esta sección de la defensa vemos alguna que otra métrica nueva que nos aporta información diferente cuando hablamos de equipo.
    1. **Robos, tapones y kills**. En esta primera sección no modificamos los gráficos, simplemente añadimos una nueva métrica, Kills, la cual contabiliza paradas consecutivas del ataque rival. Así pues, tres paradas consecutivas equivalen a una kill (una parada quiere decir que el rival ha finalizado su posesión de ataque sin anotar)
    2. **Detallando los tapones**. Porcentaje de tapones por contexto de tiro.
    3. **Disciplina defensiva**. Al contar con datos del equipo entero de toda la temporada, creo que puede ser más útil ver las faltas provocadas y recibidas por partido, más allá que las normalizadas a 100 posesiones.
    4. **Faltas de tiro: forzadas vs concedidas**. De nuevo, el título lo dice todo, se trata de una comparativa de faltas de tiro forzadas y concedidas. Estas dos métricas, PSF FREQ y DSF FREQ, frecuencia de faltas de tiro concedidas y forzadas respectivamente no son más que el ratio de faltas de tiro cometidas (forzadas) y faltas totales cometidas (forzadas).
    5. **Faltas de tiro forzadas vs FT Rate**. Al igual que con el gráfico de burbujas de las posesiones, con este gráfico de dispersión se puede debatir largo y tendido, ya que a la pregunta de ¿es una buena opción jugar a forzar faltas de tiro? La respuesta no es tan sencilla como parece. Así pues, este gráfico intenta aportar algo de luz al asunto, comparando la frecuencia de faltas de tiro forzadas con el ratio de tiros libres.
6. **Impacto & Avanzadas**. Esta sección es la que mayor cambio recibe, ya que al contar con las estadísticas globales, estas se reducen mucho, centrándonos así en los ratings, tanto ofensivo como defensivo y por supuesto el net rating.

1. **OFF vs DEF Ratings**. El mismo gráfico que teníamos con los ratings individuales ahora lo vemos con los ratings del equipo, creando cuatro cuadrantes que cumplen las mismas condiciones que el gráfico anterior.
2. **Descomposición Net Rating**. Con gráficos de cascada vemos de manera muy visual la descomposición del net rating, donde el rating ofensivo suma y el defensivo resta. De igual manera, nos sirve a modo de comparativa rápida en caso de así desearlo.
3. **Resumen rápido**. El no poder contar con el PER, únicamente veremos la VAL y el +/- en este último gráfico de barras.

3.3.2. Capturas de pantalla

De manera similar a Player Report, veremos unos pocos gráficos de una comparativa de dos equipos. En este caso, estamos comparando a los dos finalistas de este último año, Fenerbahce Beko Istanbul y AS Monaco.

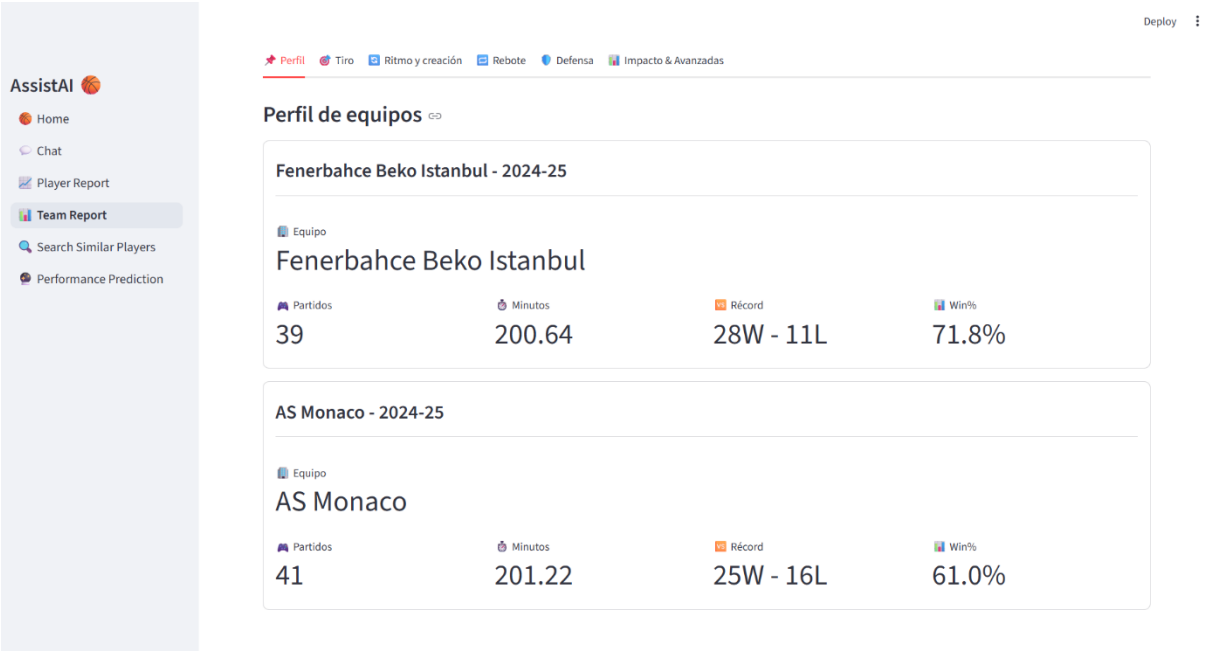

Figura 15. AssistAI - Team Report - Perfil de equipos

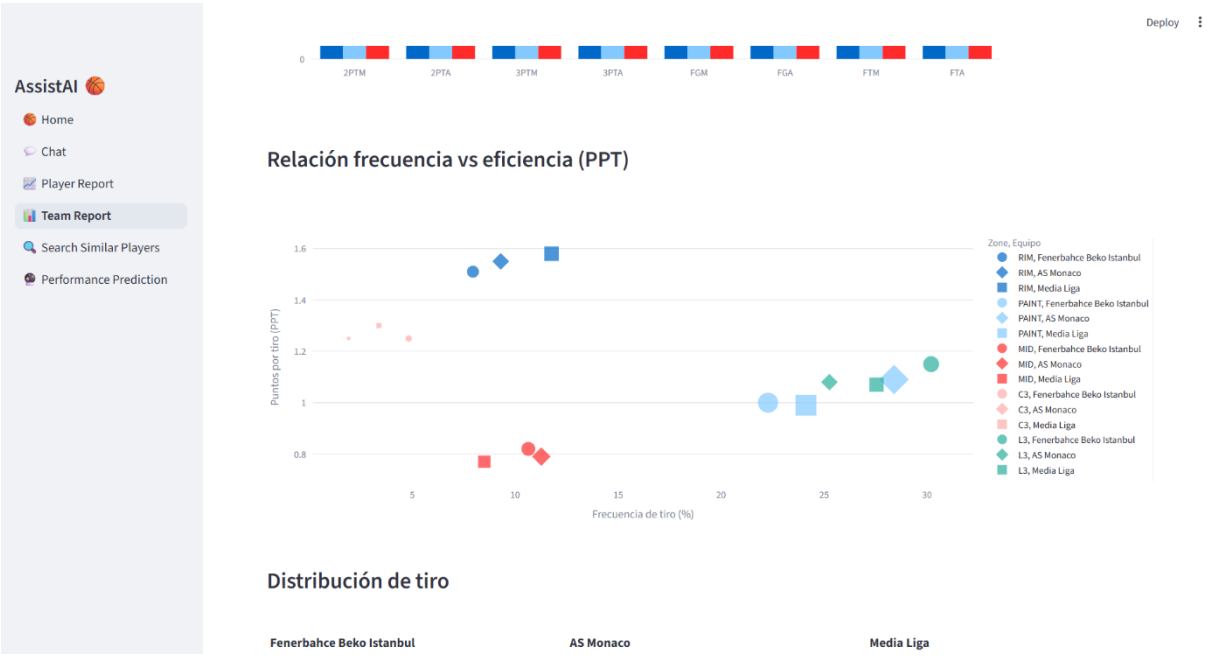

Figura 16. AssistAI - Team Report - Frecuencia de tiro por posición vs PPT

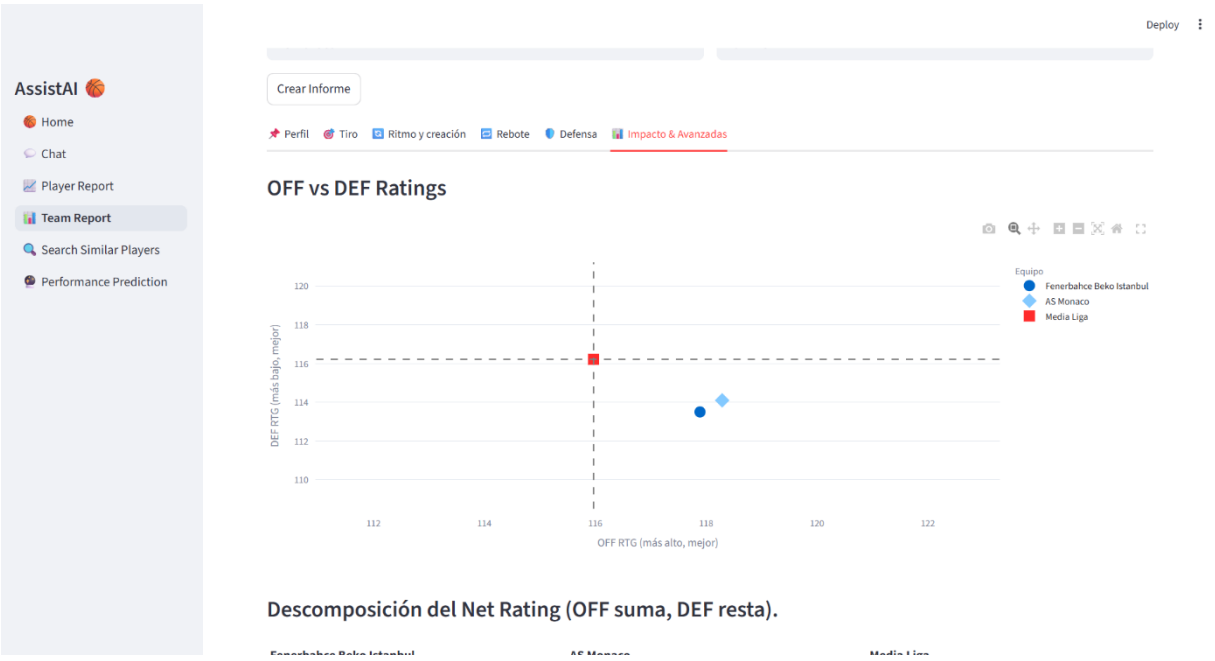

Figura 17. AssistAI - Team Report - OFF vs DEF Ratings

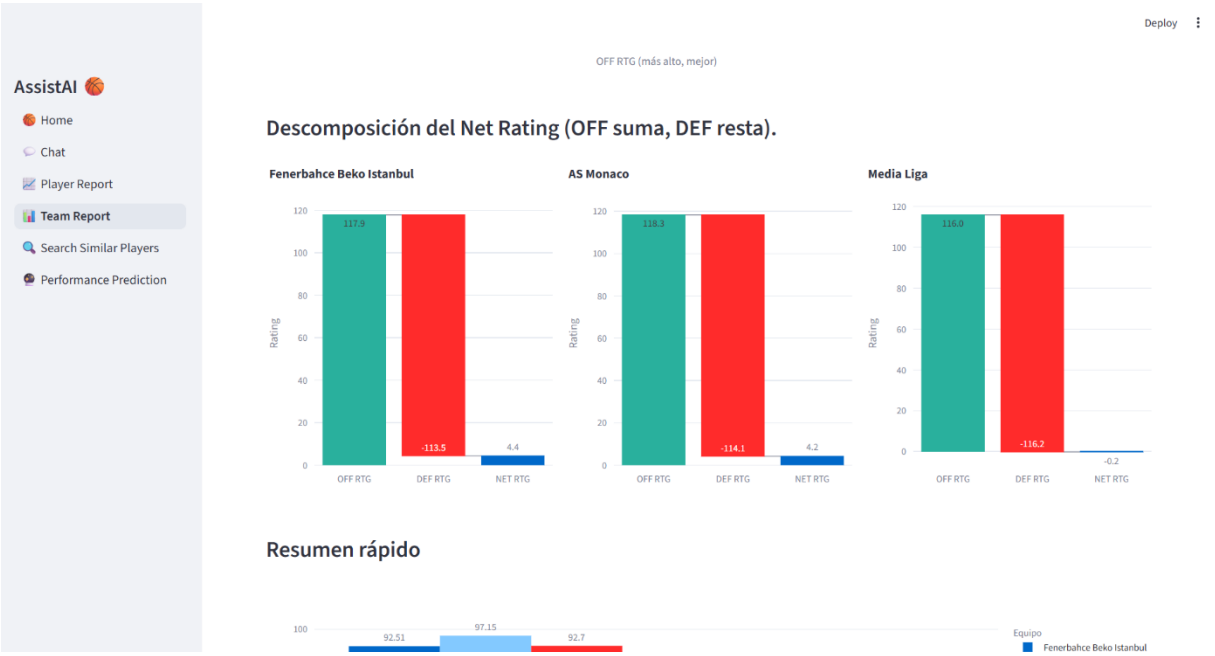

Figura 18. AssistAI - Team Report - Descomposición del Net Rating

3.4. Search Similar Players

La funcionalidad de búsqueda de jugadores similares, expuesta a través del endpoint `/search-similar` en el backend y su interfaz en el frontend, es una herramienta poderosa para el scouting y el análisis de perfiles de jugadores. Permite a los usuarios encontrar jugadores con características estadísticas parecidas a un jugador de referencia, lo cual es invaluable para identificar talentos, buscar reemplazos o analizar el estilo de juego de un rival.

3.4.1. Búsqueda

El proceso de búsqueda de jugadores similares se basa en el cálculo de la **similitud coseno** entre los vectores de estadísticas de los jugadores. Aquí explico cómo funciona y las decisiones clave tomadas:

1. **Selección del jugador de referencia**. El usuario selecciona un `player_name` y una `season_id` como punto de partida. El backend (`src/backend/search_similar/main.py`) recupera todas las estadísticas de ese jugador para la temporada especificada desde la tabla `player_stats` de la base de datos.

2. **Recuperación de jugadores para comparación**. Se recuperan las estadísticas de todos los demás jugadores en la base de datos.
    - **Filtro por rol (opcional)**. El usuario puede activar un filtro (`same_role=True`) para comparar el jugador de referencia solo con otros jugadores que compartan su misma posición (`role`). Esto es crucial, ya que comparar un base con un pívot, por ejemplo, rara vez arrojaría resultados significativos debido a las diferencias inherentes en sus roles y estadísticas.
3. **Selección de Métricas Relevantes**: No todas las columnas de la tabla `player_stats` son útiles para la comparación de similitud. Se excluyen columnas identificativas (`id`, `name`, `tm_name`, `season_id`, `nat`) y métricas que no reflejan directamente el estilo de juego o el rendimiento intrínseco (gp, `w`, `l`, `w_pct`). Las métricas restantes (`metrics` en `src/backend/search_similar/main.py`) son las que se utilizan para construir los vectores de comparación.
4. **Estandarización de Datos**: Las estadísticas de baloncesto tienen diferentes escalas (por ejemplo, los puntos pueden ser 20, mientras que los porcentajes son 0.5). Para que todas las métricas contribuyan equitativamente a la similitud, se aplica una **estandarización** utilizando `StandardScaler` de `sklearn.preprocessing`. Esto transforma los datos para que tengan una media de 0 y una desviación estándar de 1.
5. **Aplicación de Pesos (Opcional)**: Esta es una característica avanzada y muy potente. El usuario puede proporcionar un diccionario de `weights` (pesos) para dar mayor o menor importancia a ciertas estadísticas. Por ejemplo, un entrenador podría estar buscando un tirador y querer dar más peso a `three_pt_pct` o `pts`.
    - **Implementación**: Si se proporcionan pesos, el array de estadísticas estandarizadas se multiplica por un `weight_array` donde cada elemento corresponde al peso asignado a la métrica. Las métricas no especificadas en los pesos reciben un peso por defecto de 1.0. Esto permite una búsqueda altamente personalizada.
6. **Cálculo de Similitud Coseno**: La similitud coseno es una métrica que mide el coseno del ángulo entre dos vectores. Un valor cercano a 1 indica alta similitud (los vectores apuntan en la misma dirección), mientras que un valor cercano a 0 indica baja similitud (los vectores son casi ortogonales). Se calcula la similitud coseno entre el vector del jugador de referencia y el vector de cada uno de los demás jugadores.

7. **Resultados**: Los jugadores se ordenan por su `similarity_score` (puntuación de similitud) de mayor a menor. El endpoint devuelve los 10 jugadores más similares, junto con su nombre, temporada, equipo y la puntuación de similitud.

Este enfoque permite una búsqueda flexible y precisa, adaptándose a las necesidades específicas del usuario y proporcionando insights valiosos para la gestión de plantillas y el análisis de talento.

3.4.2. Fragmentos de código

`src/backend/search_similar/main.py`. Lógica principal del endpoint de búsqueda de jugadores:

```
from fastapi import APIRouter, HTTPException, Query
from typing import Optional, Dict
import psycopg2
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import cosine_similarity
from config import ADVANCED_DB_CONFIG

router = APIRouter()

@router.get("/compare-player")
def compare_player(
    player_name: str = Query(..., description="Nombre del jugador a comparar"),
    season_id: str = Query(..., description="Temporada del jugador a comparar"),
    same_role: bool = Query(False, description="Comparar solo con jugadores del mismo rol"),
    weights: Optional[Dict[str, float]] = None
):
    """
    Compara las estadísticas de un jugador con otros jugadores buscando similitudes.
    """
    try:
        conn = psycopg2.connect(**ADVANCED_DB_CONFIG)

        # Obtener jugador base
        base_stats_query = """
            SELECT * FROM player_stats
            WHERE name = %s AND season_id = %s
            LIMIT 1
        """
        base_df = pd.read_sql(base_stats_query, conn, params=(player_name, season_id))
        if base_df.empty:
            conn.close()
            raise HTTPException(status_code=404, detail=f"Jugador '{player_name}' no encontrado
en la temporada '{season_id}'.")

        base_role = base_df.iloc[0]['role']

        # Query dinámico
        query_parts = ["(s.name != %s OR s.season_id != %s)"]
        params = [player_name, season_id]
        if same_role:
            query_parts.append("s.role = %s")
            params.append(base_role)

        base_query = f"""
        SELECT * FROM player_stats s
```

```
        WHERE {' AND '.join(query_parts)}
        """
        df = pd.read_sql(base_query, conn, params=params)

        conn.close()
        if df.empty:
            raise HTTPException(status_code=404, detail="No se encontraron jugadores para
comparar.")

        # Detectar columnas numéricas
        numeric_cols = base_df.select_dtypes(include=[np.number]).columns.tolist()
        exclude_cols = ['id', 'name', 'tm_name', 'season_id', 'nat', 'gp', 'w', 'l', 'w_pct']
        metrics = [col for col in numeric_cols if col not in exclude_cols]

        # Construir dataframe completo
        full_df = pd.concat([base_df[metrics], df[metrics]], ignore_index=True)

        # Estandarizar
        scaler = StandardScaler()
        scaled = scaler.fit_transform(full_df)

        # Aplicar pesos
        if weights:
            weight_array = np.array([weights.get(col, 1.0) for col in metrics])
            scaled = scaled * weight_array

        # Calcular similitud coseno
        sim_matrix = cosine_similarity(scaled)
        base_similarities = sim_matrix[0, 1:]

        # Construir resultado
        similarities = []
        for i, sim in enumerate(base_similarities):
            similarities.append({
                "name": df.iloc[i]['name'],
                "season_id": df.iloc[i]['season_id'],
                "team_name": df.iloc[i]['tm_name'],
                "similarity_score": round(sim * 100, 2)
            })

        similarities.sort(key=lambda x: x['similarity_score'], reverse=True)

        return {
            "jugador_base": {"name": player_name, "season_id": season_id, "role": base_role},
            "mismo_rol": same_role,
            "resultados_similares": similarities[:10]
        }

    except psycopg2.Error as db_error:
        raise HTTPException(status_code=500, detail=f"Error de base de datos: {db_error}")
    except Exception:
        raise HTTPException(status_code=500, detail="Error inesperado")
```

3.4.3. Capturas de pantalla

Una búsqueda normal se verá como en estas dos próximas imágenes, donde buscamos un reemplazo para Jabari Parker, usando sus estadísticas de este último año, buscando en su misma posición y con varios pesos, dándole mucha importancia a los puntos y puntos por posesión y reduciéndosela al porcentaje de pérdidas.

Figura 19. AssistAI - Búsqueda Jugadores Similares - Selección

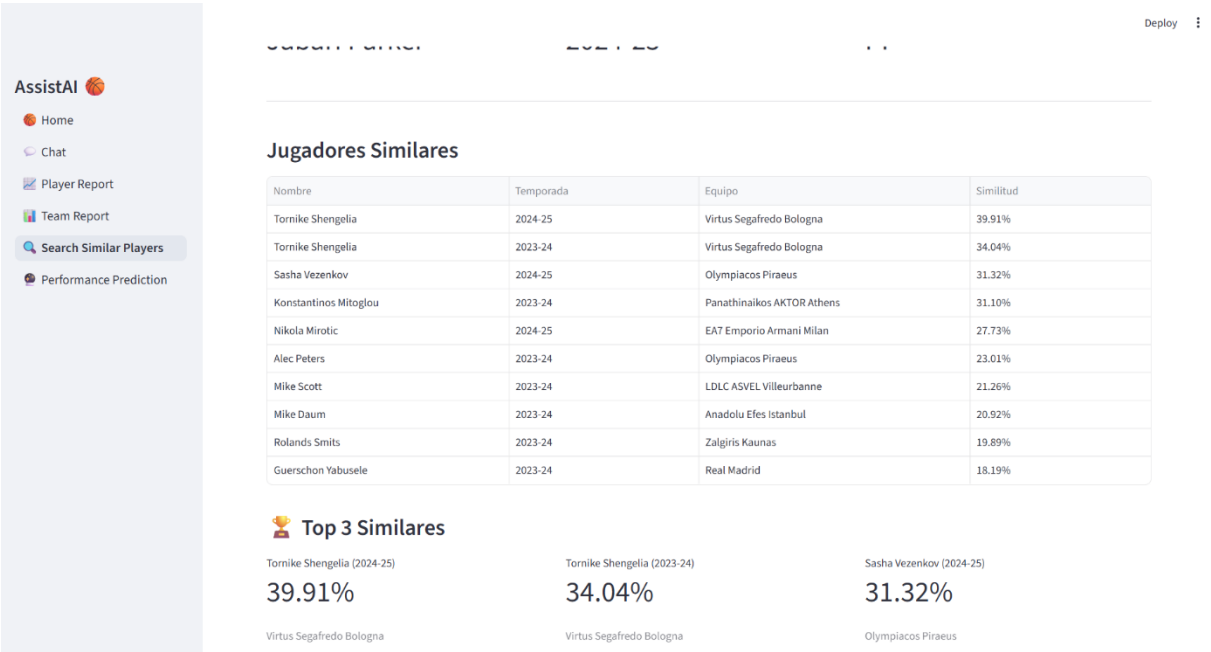

Figura 20. AssistAI - Búsqueda Jugadores Similares - Resultado

3.5. Performance Prediction

La funcionalidad de Performance Prediction, accesible a través del endpoint `/performance-prediction` en el backend y su página correspondiente en el frontend, es una de las

características más avanzadas y estratégicas de la plataforma. Permite a los usuarios estimar el rendimiento colectivo (Net Rating) de un quinteto de jugadores seleccionado, basándose en sus estadísticas individuales.

3.5.1. Modelo y proceso de entrenamiento

El corazón de esta funcionalidad es un modelo predictivo entrenado con datos estadísticos de quintetos reales. El proceso de desarrollo del modelo, documentado en `src/backend/performance_prediction/PerformancePredictor-training.ipynb`, fue un camino de experimentación y refinamiento.

Datos de Entrenamiento: Los datos para el entrenamiento se extrajeron de `hackastat.eu`, específicamente de los `lineups` (quintetos) registrados partido a partido para las temporadas 23/24 y 24/25. Cada fila de datos representaba un quinteto que había jugado un cierto número de minutos y posesiones, junto con sus estadísticas como quinteto. No contaba con la estadística de Net Rating, por lo que tuvimos que calcularla con los datos con los que contábamos antes de construir el dataset de entrenamiento para poder usarla como variable objetivo.

A la hora de crear el dataset tuvimos que tratar los datos de manera aislada para obtener un DataFrame con las estadísticas de los jugadores de manera individual, así como la variable objetivo. Tuvo que hacerse de manera aislada la 23/24 con la 24/25 para no mezclar estadísticas de diferentes temporadas y modificar el resultado final.

Por otro lado, siento que fueron pocos datos para el dataset de entrenamiento, sin embargo, en `hackastat.eu` estas son todas las estadísticas de la Euroliga que cumplen los requisitos que necesitamos.

Proceso de Prueba y Error en la Selección de Features: La representación de un quinteto para el modelo fue el mayor desafío y el foco de varias iteraciones, tras haber creado el dataset de entrenamiento y el esqueleto de un script capaz de crear modelos, únicamente quedaba iterar para encontrar una buena opción con la que poder tener un buen modelo.

- **v1 – ALL IND (todas las estadísticas individuales)**
    - **Enfoque**: Inicialmente, se intentó utilizar todas las estadísticas individuales de cada uno de los cinco jugadores de manera aislada. Esto significaba que, si un jugador tenía 100 estadísticas, un quinteto se representaría con 500 features.

- **Resultados**:
  - Baseline (media ponderada) -> MAE: 30.019 | RMSE: 37.619 | R^2: -0.001
  - CV RMSE (Ridge): 52.602
  - CV RMSE (HGBR): 38.643
  - Test (HGBR) -> MAE: 30.483 | RMSE: 37.868 | R^2: -0.014
- **Análisis**: Los resultados fueron muy pobres. El modelo no lograba predecir de manera correcta, y el R^2 negativo en el test indicaba que el modelo era peor que simplemente predecir la media. Esto se atribuyó a la **maldición de la dimensionalidad**, demasiadas características (features) para el tamaño del dataset, lo que llevaba a un sobreajuste y una pobre generalización. Además, el orden de los jugadores en el quinteto afectaba la representación, lo cual no era deseable (un quinteto A-B-C-D-E debería ser igual a E-D-C-B-A).

- **v2 – SUM + MEAN (suma y media de las estadísticas individuales)**
  - **Enfoque**: Para reducir la dimensionalidad y hacer la representación independiente del orden, se decidió agregar las estadísticas individuales de los cinco jugadores. Para cada estadística (ej. PTS), se crearon dos nuevas features: `PTS_sum` (suma de los puntos de los 5 jugadores) y `PTS_mean` (media de los puntos de los 5 jugadores). Esto redujo drásticamente el número de features (de 500 a 200-300, dependiendo de las estadísticas disponibles).
  - **Resultados**:
    - Baseline (media ponderada) -> MAE: 29.866 | RMSE: 37.265 | R^2: -0.002
    - CV RMSE (Ridge): 41.908
    - CV RMSE (HGBR): 37.429
    - Test (HGBR) -> MAE: 28.844 | RMSE: 36.437 | R^2: 0.042
  - **Análisis**: Se observa una mejora significativa. El R^2 pasó a ser positivo, indicando que el modelo tenía cierta capacidad predictiva. El RMSE y MAE también mejoraron. Esta representación capturaba la “esencia” colectiva del quinteto sin depender del orden.
- **v3 – MEAN (media de estadísticas)**
  - **Enfoque:** Se probó una variante de la v2, utilizando solo la media de las estadísticas individuales, eliminando las sumas.
  - **Resultados:**

        - Baseline (media ponderada) -> MAE: 29.866 | RMSE: 37.265 | R^2: -0.002
        - CV RMSE (Ridge): 40.923
        - CV RMSE (HGBR): 37.515
        - Test (HGBR) -> MAE: 29.058 | RMSE: 36.541 | R^2: 0.037
    - **Análisis**: Los resultados fueron muy similares a la v2, ligeramente peores en el test. Esto sugería que la suma también aportaba información valiosa sobre el volumen total de producción del quinteto.
- **v4 – ALL IND + ORDER (todas las estadísticas individuales, pero con orden)**
    - **Enfoque**: Se intentó volver a las estadísticas individuales, pero esta vez asignando una posición fija a cada jugador dentro del quinteto (ej. Jugador 1 siempre es el base, Jugador 2 el escolta, etc.). Esto requería un preprocesamiento más complejo para asegurar que los jugadores se asignaran consistentemente a las "posiciones" de features.
    - **Resultados**:
        - Baseline (media ponderada) -> MAE: 29.866 | RMSE: 37.265 | R^2: -0.002
        - CV RMSE (Ridge): 52.448
        - CV RMSE (HGBR): 38.205
        - Test (HGBR) -> MAE: 29.995 | RMSE: 37.197 | R^2: 0.002
    - **Análisis**: Los resultados volvieron a ser pobres, similares a la v1. Esto confirmó que la alta dimensionalidad y la escasez de datos para entrenar patrones complejos entre jugadores individuales seguía siendo un problema, incluso con un orden fijo.

Modelo Final Seleccionado: Tras estas iteraciones, la **versión 2 (SUM + MEAN)** fue la que ofreció el mejor equilibrio entre rendimiento predictivo y robustez. El modelo específico utilizado fue **HistGradientBoostingRegressor (HGBR)**, que consistentemente superó a Ridge Regression en las pruebas de validación cruzada y en el conjunto de test.

El pipeline del modelo incluye: * `SimpleImputer(strategy='median')`: Para manejar cualquier valor nulo en las estadísticas (aunque se hizo un esfuerzo por tener datos completos). * `HistGradientBoostingRegressor`: El algoritmo de boosting, configurado con `max_iter=500`, `learning_rate=0.05`, y `early_stopping` para evitar el sobreajuste.

El modelo entrenado (`performance_model.pkl`) y las columnas de features esperadas (`feature_columns.json`) se guardan y se cargan en el backend (`src/backend/performance_prediction/main.py`) para realizar las predicciones en tiempo real. Cuando el usuario selecciona un quinteto, el backend recupera las estadísticas individuales de cada jugador, calcula las sumas y medias de las features relevantes, construye el vector de entrada en el formato exacto que espera el modelo y realiza la predicción del Net Rating.

Este enfoque, aunque no perfecto (el R^2 de 0.042 indica que aún hay mucho margen de mejora y que el Net Rating es una métrica difícil de predecir con solo estadísticas individuales), representa un avance significativo respecto a la línea base y ofrece una herramienta experimental valiosa para la toma de decisiones tácticas.

3.5.2. Fragmentos de código

`src/backend/performance_prediction/PerformancePredictor-training.ipynb`.

Fichero usado para la creación y el entrenamiento del modelo que posteriormente usaremos (ejecutado desde Google Colab, lo he modificado para una correcta lectura:

```
# In[8]:
import os
import json
import numpy as np
import pandas as pd

from pathlib import Path
from typing import List

from sklearn.model_selection import KFold, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.inspection import permutation_importance
import joblib

from google.colab import drive
drive.mount('/content/drive')

RANDOM_STATE = 42

DATA_DIR = Path('/content/drive/MyDrive/Máster Basket Data Analytics/12. TFM/data')
LINEUPS_2324 = DATA_DIR / '23-24_Lineups_GbG.csv'
LINEUPS_2425 = DATA_DIR / '24-25_Lineups_GbG.csv'
PLAYERS_2324 = DATA_DIR / '23-24_Players.csv'
PLAYERS_2425 = DATA_DIR / '24-25_Players.csv'

MIN_MINUTES = 10.0
MIN_POSSESSIONS = 10.0
WEIGHT_BY = 'poss'
```

```
MODEL_PATH = Path('best_model.pkl')
FEATURES_PATH = Path('feature_columns.json')

# In[3]:
def read_csv_safely(path: Path, sep=';') -> pd.DataFrame:
    return pd.read_csv(path, sep=sep)

def compute_net_rating(df: pd.DataFrame) -> pd.Series:
    poss = (df.get("POSS", 0).fillna(0) + df.get("OPP POSS", 0).fillna(0)) / 2.0
    net = (df.get("PTS", 0).fillna(0) - df.get("OPP PTS", 0).fillna(0))
    return np.where(poss > 0, 100.0 * net / poss, np.nan)

def filter_lineups(df: pd.DataFrame, min_min: float, min_poss: float) -> pd.DataFrame:
    poss = (df.get("POSS", 0).fillna(0) + df.get("OPP POSS", 0).fillna(0))
    cond = (df.get("MIN", 0).fillna(0) >= min_min) & (poss >= min_poss)
    return df.loc[cond].copy()

def promedio_stats(df_players: pd.DataFrame) -> pd.DataFrame:
    num_cols = df_players.select_dtypes(include='number').columns.tolist()
    non_num = [c for c in df_players.columns if c not in num_cols]
    if 'NAME' in non_num:
        non_num.remove('NAME')
    agg = {c: 'mean' for c in num_cols}
    agg.update({c: 'first' for c in non_num})
    return df_players.groupby('NAME', as_index=False).agg(agg)

def sanitize_commas_to_dots(df: pd.DataFrame, columns: List[str]) -> pd.DataFrame:
    for c in columns:
        if df[c].dtype == object:
            df[c] = (df[c].astype(str)
                          .str.replace(',', '.', regex=True)
                          .replace(['-', 'nan', 'None', ''], np.nan))
        df[c] = pd.to_numeric(df[c], errors='coerce')
    return df

def merge_lineups_players(df_lineups: pd.DataFrame, df_players: pd.DataFrame) -> pd.DataFrame:
    keep_cols = [c for c in df_lineups.columns if c in ['PL 1','PL 2','PL 3','PL 4','PL
5','PTS','OPP PTS','POSS','OPP POSS','MIN']]
    if 'NET_RTG' in df_lineups.columns:
        keep_cols.append('NET_RTG')
    base = df_lineups[keep_cols].copy()

    stat_cols = [c for c in df_players.columns if c != 'NAME']
    out_rows = []

    for idx, row in base.iterrows():
        players = [row[f'PL {i}'] for i in range(1,6)]
        df_sub = df_players[df_players['NAME'].isin(players)]

        agg = {}
        for stat in stat_cols:
            vals = df_sub[stat].values
            agg[f"{stat}_sum"] = np.nansum(vals)
            agg[f"{stat}_mean"] = np.nanmean(vals)

        agg_row = row.to_dict()
        agg_row.update(agg)
        out_rows.append(agg_row)

    return pd.DataFrame(out_rows)

# In[4]:
lineups_2324 = read_csv_safely(LINEUPS_2324)
lineups_2425 = read_csv_safely(LINEUPS_2425)
players_2324 = read_csv_safely(PLAYERS_2324)
players_2425 = read_csv_safely(PLAYERS_2425)

print('Lineups 23/24:', lineups_2324.shape, '| Players 23/24:', players_2324.shape)
print('Lineups 24/25:', lineups_2425.shape, '| Players 24/25:', players_2425.shape)
```

```
for df in [lineups_2324, lineups_2425]:
    for col in ["MIN", "POSS", "OPP POSS"]:
        df[col] = df[col].astype(str).str.replace(",", ".")
        df[col] = pd.to_numeric(df[col], errors="coerce")

if 'NET_RTG' not in lineups_2324.columns:
    lineups_2324['NET_RTG'] = compute_net_rating(lineups_2324)
if 'NET_RTG' not in lineups_2425.columns:
    lineups_2425['NET_RTG'] = compute_net_rating(lineups_2425)

lineups_2324_f = filter_lineups(lineups_2324, MIN_MINUTES, MIN_POSSESSIONS)
lineups_2425_f = filter_lineups(lineups_2425, MIN_MINUTES, MIN_POSSESSIONS)

print('Filtrados -> 23/24:', lineups_2324_f.shape, ' | 24/25:', lineups_2425_f.shape)

# In[5]:
def guess_numeric_like_columns(df: pd.DataFrame) -> List[str]:
    cand = []
    for c in df.columns:
        if c == 'NAME':
            continue
        if pd.api.types.is_numeric_dtype(df[c]):
            cand.append(c)
        elif pd.api.types.is_object_dtype(df[c]):
            sample = df[c].astype(str).head(50).str.contains(r'^[\d,\.\-]+$', regex=True).mean()
            if sample > 0.6:
                cand.append(c)
    return cand

player_stat_cols_2324 = guess_numeric_like_columns(players_2324)
player_stat_cols_2425 = guess_numeric_like_columns(players_2425)
common_player_stats =
sorted(list(set(player_stat_cols_2324).intersection(set(player_stat_cols_2425))))

players_2324_keep = ['NAME'] + common_player_stats
players_2425_keep = ['NAME'] + common_player_stats
players_2324_f = sanitize_commas_to_dots(players_2324[players_2324_keep].copy(),
common_player_stats)
players_2425_f = sanitize_commas_to_dots(players_2425[players_2425_keep].copy(),
common_player_stats)

players_2324_f = promedio_stats(players_2324_f)
players_2425_f = promedio_stats(players_2425_f)

print('Player stat columns (common):', len(common_player_stats))

# In[6]:
df_2324 = merge_lineups_players(lineups_2324_f, players_2324_f)
df_2324['SEASON'] = '23-24'
df_2425 = merge_lineups_players(lineups_2425_f, players_2425_f)
df_2425['SEASON'] = '24-25'

df_all = pd.concat([df_2324, df_2425], axis=0, ignore_index=True)
num_cols_all = df_all.select_dtypes(include='number').columns.tolist()
df_all[num_cols_all] = df_all[num_cols_all].replace([np.inf, -np.inf], np.nan)

print('Tabla final:', df_all.shape)
df_all.head(2)

# In[7]:
target_col = 'NET_RTG'
feature_cols = [c for c in df_all.columns if c.endswith('_sum') or c.endswith('_mean')]

def compute_sample_weights(df: pd.DataFrame, weight_by: str = 'poss') -> np.ndarray:
    if weight_by == 'poss' and all(col in df.columns for col in ['POSS','OPP POSS']):
        poss = (df['POSS'].fillna(0) + df['OPP POSS'].fillna(0)) / 2.0
        w = poss.clip(lower=1.0)
    elif weight_by == 'min' and 'MIN' in df.columns:
        w = df['MIN'].fillna(0).clip(lower=1.0)
    else:
```

```
        w = pd.Series(1.0, index=df.index)
    return w.to_numpy(dtype=float)

train_idx = df_all['SEASON'] == '23-24'
test_idx  = df_all['SEASON'] == '24-25'

X_train = df_all.loc[train_idx, feature_cols].copy()
y_train = df_all.loc[train_idx, target_col].copy()
w_train = compute_sample_weights(df_all.loc[train_idx], WEIGHT_BY)

X_test  = df_all.loc[test_idx, feature_cols].copy()
y_test  = df_all.loc[test_idx, target_col].copy()
w_test  = compute_sample_weights(df_all.loc[test_idx], WEIGHT_BY)

print('X_train:', X_train.shape, '| X_test:', X_test.shape)

# In[9]:
def evaluate(y_true, y_pred, sample_weight=None, label=''):
    mae = mean_absolute_error(y_true, y_pred, sample_weight=sample_weight)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred, sample_weight=sample_weight))
    r2 = r2_score(y_true, y_pred, sample_weight=sample_weight)
    print(f'{label} -> MAE: {mae:.3f} | RMSE: {rmse:.3f} | R²: {r2:.3f}')
    return {'mae': mae, 'rmse': rmse, 'r2': r2}

baseline_value = np.average(y_train.to_numpy(), weights=w_train) if len(y_train) > 0 else
float(y_train.mean())
baseline_pred = np.full_like(y_test, baseline_value, dtype=float)
baseline_metrics = evaluate(y_test, baseline_pred, sample_weight=w_test, label='Baseline (media
ponderada)')

ridge_pipe = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler()),
    ('model', Ridge(random_state=RANDOM_STATE, alpha=10.0))
])

hgb_pipe = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('model', HistGradientBoostingRegressor(
        random_state=RANDOM_STATE,
        max_iter=500,
        learning_rate=0.05,
        early_stopping=True,
        validation_fraction=0.15,
        min_samples_leaf=20
    ))
])

n_splits = 5 if len(X_train) >= 200 else 3
cv = KFold(n_splits=n_splits)

def cv_rmse(pipe, X, y):
    scores = cross_val_score(pipe, X, y, cv=cv,
                             scoring='neg_root_mean_squared_error')
    return -scores.mean()

models = [('Ridge', ridge_pipe), ('HGBR', hgb_pipe)]
cv_results = {}
for name, pipe in models:
    mean_rmse = cv_rmse(pipe, X_train, y_train)
    cv_results[name] = mean_rmse
    print(f'CV RMSE ({name}): {mean_rmse:.3f}')

best_name = min(cv_results, key=cv_results.get)
best_pipe = dict(models)[best_name]

best_pipe.fit(X_train, y_train, model__sample_weight=w_train if 'model__sample_weight' in
best_pipe.get_params() else None)

yhat_test = best_pipe.predict(X_test)
```

```
test_metrics = evaluate(y_test, yhat_test, sample_weight=w_test, label=f'Test ({best_name})')

# In[10]:
try:
    perm = permutation_importance(best_pipe, X_test, y_test, n_repeats=10,
random_state=RANDOM_STATE, scoring='neg_root_mean_squared_error')
    importances = pd.DataFrame({'feature': feature_cols, 'importance':
perm.importances_mean}).sort_values('importance', ascending=False)
    display(importances.head(20))
except Exception as e:
    print('Permutation importance no disponible:', e)

joblib.dump(best_pipe, MODEL_PATH)
with open(FEATURES_PATH, 'w', encoding='utf-8') as f:
    json.dump({'feature_columns': feature_cols, 'best_model': best_name}, f, ensure_ascii=False,
indent=2)

print(f'Modelo guardado en: {MODEL_PATH.resolve()}')
print(f'Columnas guardadas en: {FEATURES_PATH.resolve()}')
```

src/backend/performance_prediction/main.py. Fichero que contiene la lógica del endpoint de predicción, haciendo uso del modelo entrenado:

```
import os
import json
import pandas as pd
import psycopg2
import psycopg2.extras
import joblib
from fastapi import APIRouter, HTTPException
from typing import List, Dict
import numpy as np
from config import ADVANCED_DB_CONFIG, FEATURE_TO_DB

router = APIRouter()

BASE_DIR = os.path.dirname(__file__)
MODEL_PATH = os.path.join(BASE_DIR, "performance_model.pkl")
FEATURE_COLUMNS_JSON = os.path.join(BASE_DIR, "feature_columns.json")

def load_feature_columns() -> List[str]:
    if not os.path.exists(FEATURE_COLUMNS_JSON):
        raise FileNotFoundError(f"feature_columns.json no encontrado en: {FEATURE_COLUMNS_JSON}")
    with open(FEATURE_COLUMNS_JSON, "r", encoding="utf-8") as f:
        payload = json.load(f)
    cols = payload.get("feature_columns")
    if not isinstance(cols, list) or not cols:
        raise ValueError("feature_columns.json no contiene una lista 'feature_columns' válida.")
    return cols

def load_model():
    if not os.path.exists(MODEL_PATH):
        raise FileNotFoundError(f"Model file not found at: {MODEL_PATH}")
    model = joblib.load(MODEL_PATH)
    print("Modelo cargado correctamente.")
    return model

def safe_num(x) -> float:
    """
    Convierte a float y sustituye None/NaN por 0.0
    """
    try:
        v = float(x)
        if np.isnan(v):
```

```
            return 0.0
        return v
    except Exception:
        return 0.0

try:
    predictor_model = load_model()
    print("Modelo de predicción cargado correctamente.")
except Exception as e:
    print("Error loading model")
    predictor_model = None

try:
    FEATURE_COLUMNS = load_feature_columns()
    print(f"feature_columns.json cargado ({len(FEATURE_COLUMNS)} columnas).")
except Exception as e:
    print("Error loading feature columns")
    FEATURE_COLUMNS = None

def get_player_stats_from_db(player_name: str, season_id: str) -> Dict:
    """
    Recupera las estadísticas de un jugador de la base de datos avanzada.
    """
    conn = None
    try:
        conn = psycopg2.connect(**ADVANCED_DB_CONFIG)
        cur = conn.cursor(cursor_factory=psycopg2.extras.DictCursor)

        sql_query = """
            SELECT *
            FROM player_stats
            WHERE name = %s AND season_id = %s;
        """
        cur.execute(sql_query, (player_name, season_id))
        stats_row = cur.fetchone()

        if stats_row:
            return dict(stats_row)
        else:
            return None
    except (Exception, psycopg2.DatabaseError) as error:
        print(f"Error al consultar la base de datos: {error}")
        return None
    finally:
        if conn is not None:
            conn.close()

@router.get("/predict")
def predict_lineup_performance(
    player1_name: str,
    season1_id: str,
    player2_name: str,
    season2_id: str,
    player3_name: str,
    season3_id: str,
    player4_name: str,
    season4_id: str,
    player5_name: str,
    season5_id: str
):
    """
    Endpoint para predecir el rendimiento (Net Rating) de un quinteto usando
    el nuevo modelo que requiere features agregadas (_sum y _mean).
    """
    if predictor_model is None:
        raise HTTPException(status_code=503, detail="El modelo de predicción no está
disponible.")
    if FEATURE_COLUMNS is None:
        raise HTTPException(status_code=503, detail="Las columnas de features no están
disponibles.")
```

```
    players_data = [
        get_player_stats_from_db(player1_name, season1_id),
        get_player_stats_from_db(player2_name, season2_id),
        get_player_stats_from_db(player3_name, season3_id),
        get_player_stats_from_db(player4_name, season4_id),
        get_player_stats_from_db(player5_name, season5_id),
    ]

    if any(p is None for p in players_data):
        raise HTTPException(
            status_code=404,
            detail="No se encontraron datos para uno o más jugadores y temporadas seleccionadas."
        )

    data_for_prediction = {col: 0.0 for col in FEATURE_COLUMNS}

    for base_name, db_col in FEATURE_TO_DB.items():
        sum_key = f"{base_name}_sum"
        mean_key = f"{base_name}_mean"
        if sum_key not in data_for_prediction and mean_key not in data_for_prediction:
            continue

        if db_col is None:
            if sum_key in data_for_prediction:
                data_for_prediction[sum_key] = 0.0
            if mean_key in data_for_prediction:
                data_for_prediction[mean_key] = 0.0
            continue

        values = [safe_num(p.get(db_col, 0.0)) for p in players_data]
        s = float(np.nansum(values))
        m = s / 5.0

        if sum_key in data_for_prediction:
            data_for_prediction[sum_key] = s
        if mean_key in data_for_prediction:
            data_for_prediction[mean_key] = m

    df_for_prediction = pd.DataFrame([data_for_prediction], columns=FEATURE_COLUMNS)

    for col in df_for_prediction.columns:
        df_for_prediction[col] = pd.to_numeric(df_for_prediction[col], errors='coerce')
    df_for_prediction = df_for_prediction.fillna(0.0)

    try:
        prediction = predictor_model.predict(df_for_prediction)
    except Exception as e:
        raise HTTPException(status_code=500, detail="Error al realizar la predicción")

    return {
        "predicted_net_rating": float(prediction[0]),
        "lineup": [p['name'] for p in players_data]
    }
```

3.5.3. Capturas de pantalla

El hacer una predicción de rendimiento en esta interfaz es muy sencillo, únicamente se han de seleccionar a los jugadores y sus temporadas y pulsar predecir. En este caso, vemos cómo queremos predecir el rendimiento que tendría el primer equipo de la Euroliga de este año.

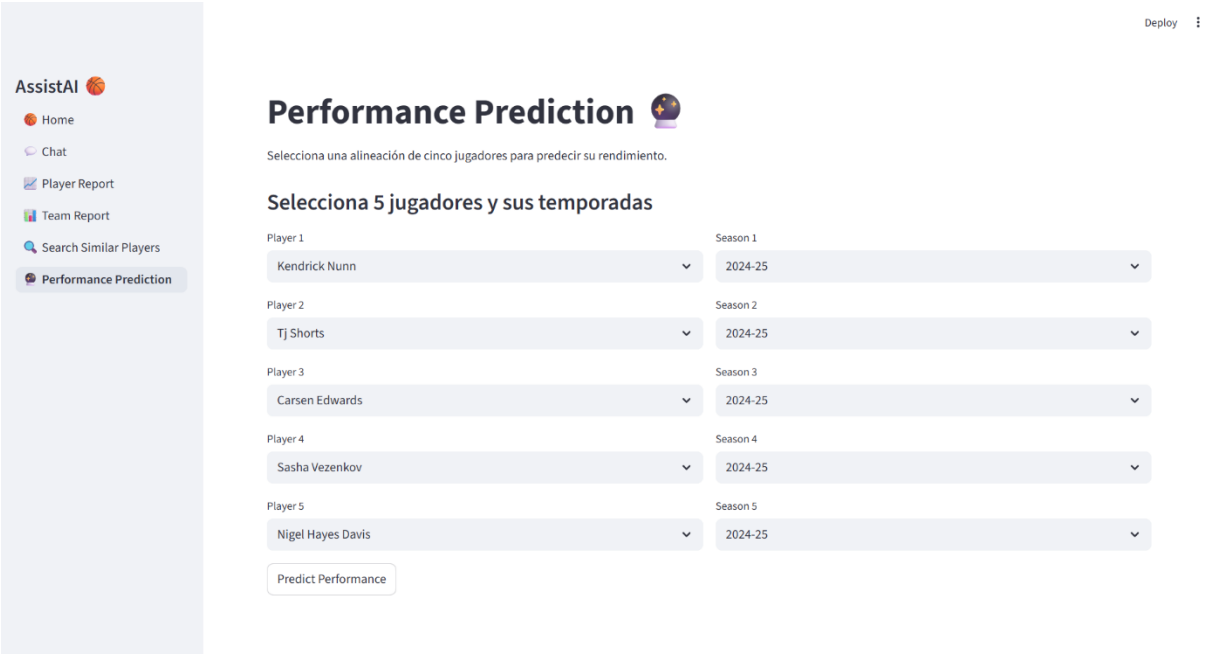

Figura 21. AssistAI - Performance Predictor - Selección

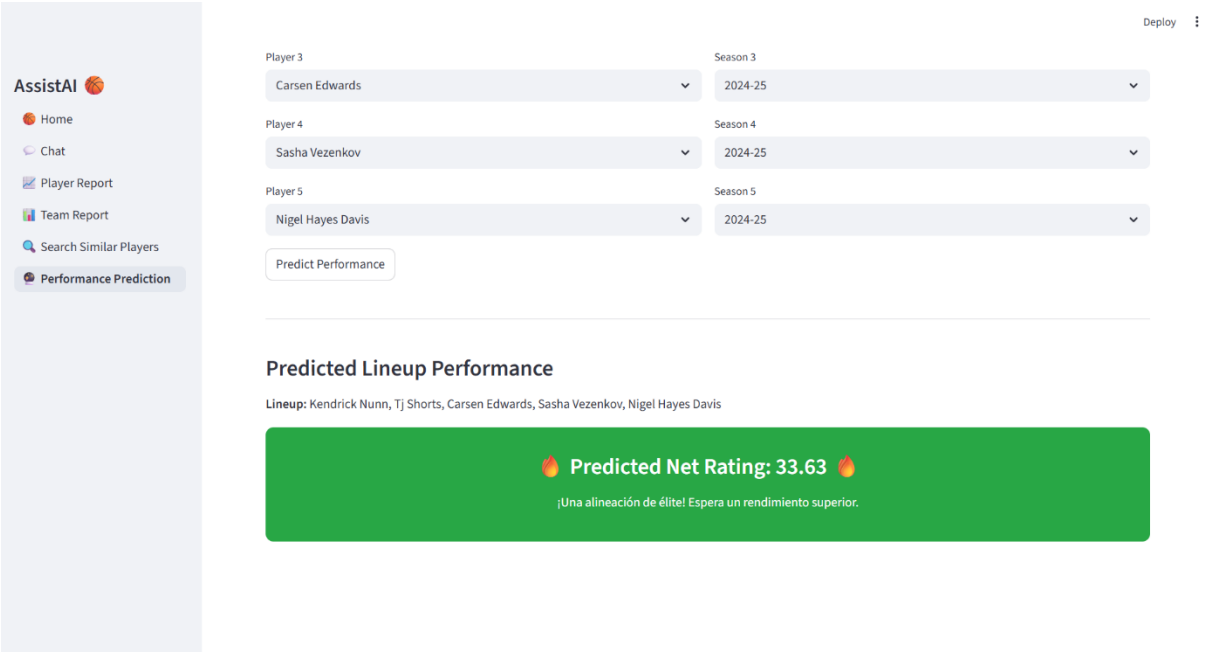

Figura 22. AssistAI - Performance Predictor - Resultado

4. Demo y ejecución

Una vez finalizada la explicación completa de la aplicación, podemos ver una demo de un uso completo de esta misma. Para ello, podemos encontrar un vídeo en https://youtu.be/CS19B3H0pqM que nos muestra lo que podría ser un uso normal de todas las funcionalidades de la aplicación a tiempo real. Cabe destacar que está corriendo todo en un portátil y como LLM para el chat está usando el GPT-4o mini de OpenAI, por lo que en otras condiciones los tiempos de carga y uso podrían variar.

La demo está dividida en el uso organizado de todas y cada una de las funciones. Estos son los tiempos de cada una de las funciones (pueden ser encontrados también en la descripción del vídeo en YouTube):

- 00:00 Página principal + selector de tema
- 00:11 Chat
- 02:08 Player Report
- 07:05 Team Report
- 10:52 Buscador Jugadores Similares
- 11:28 Predictor de rendimiento

Además de esta demo, todo el código del proyecto se encuentra en un repositorio público de GitHub: https://github.com/mapecim/AssistAI. Las instrucciones de uso las podemos encontrar en el README de este proyecto y constan únicamente de 7 pasos siendo uno de ellos opcional:

1. Clonar el repositorio.
   Para ello, git clone `https://github.com/mapecim/AssistAI.git`
2. Crear entorno virtual (opcional pero recomendado).
   Hemos de lanzar el comando `python -m venv venv`
3. Instalar dependencias. `pip install -r requirements.txt`
4. Configurar variables. En el repositorio encontrarás un archivo llamado `.env-nokeys`, el cual es una plantilla del archivo `.env` que espera AssistAI. Por eso, tendrás que renombrarlo y añadir tus claves.

5. Crear la base de datos. Para crear la base de datos que espera el proyecto únicamente hemos de ejecutar su código principal: `python etl/advanced/main.py`
6. Iniciar backend. `uvicorn src.backend.main:app --reload`
7. Iniciar frontend. `streamlit run src/frontend/app.py`

5. Lecciones aprendidas

El desarrollo de esta plataforma ha sido un viaje de aprendizaje continuo, lleno de desafíos y descubrimientos. Aquí se resumen algunas de las lecciones más importantes:

- **La importancia de la calidad y granularidad de los datos.** La transición de SportRadar a HackAStat fue una decisión crítica. Aunque implicó reducir el alcance a una sola liga, la mayor granularidad y calidad de los datos de HackAStat permitieron construir funcionalidades mucho más sofisticadas y relevantes. Es preferible tener datos profundos de una fuente fiable que datos superficiales de muchas fuentes.
- **Gestión de APIs y costos.** Las APIs gratuitas o de prueba tienen limitaciones. Es fundamental planificar el uso de la API y considerar alternativas o modelos de pago desde el inicio para evitar interrupciones en el desarrollo.
- **Diseño de esquemas de BBDD flexibles.** A medida que la comprensión de los datos y las necesidades del proyecto evolucionaron, el esquema de la base de datos tuvo que adaptarse. Un diseño inicial flexible y la capacidad de realizar migraciones eficientes son clave.
- **El desafío de la dimensionalidad en ML.** La experiencia con el modelo predictivo de quintetos fue un claro ejemplo de la "maldición de la dimensionalidad". Intentar usar todas las estadísticas individuales de cada jugador llevó a un rendimiento muy pobre. La agregación de features (sumas y medias) fue una solución efectiva para reducir la complejidad y mejorar la capacidad predictiva, demostrando que a veces "menos es más" en la ingeniería de features.
- **La complejidad de predecir el rendimiento colectivo.** El Net Rating de un quinteto es una métrica compleja influenciada por muchos factores (química, estrategia, rival, momento del partido) que no siempre se capturan en las estadísticas individuales promedio. El R^2 bajo del modelo predictivo subraya que, aunque se logró una mejora sobre la línea base, la predicción del rendimiento colectivo es un problema inherentemente difícil y multifactorial.
- **El poder de los sistemas multi-agente.** CrewAI demostró ser una herramienta excepcional para construir un chatbot versátil. La capacidad de definir roles específicos para cada agente y orquestar su colaboración simplificó enormemente la lógica del

chatbot, permitiendo manejar diferentes tipos de consultas (SQL, explicaciones, informes) de manera modular y escalable.

- **Desarrollo ágil y adaptación.** El proyecto requirió una constante adaptación a los problemas encontrados (limitaciones de API, rendimiento del modelo). Adoptar un enfoque iterativo y estar dispuesto a pivotar en las decisiones iniciales fue fundamental para el éxito.
- **Importancia de la Experiencia de Usuario (UX).** Aunque el backend es complejo, el frontend con Streamlit permitió crear una interfaz sencilla y atractiva. La facilidad de uso es crucial para que una herramienta de análisis de datos sea adoptada por usuarios no técnicos como los entrenadores

En resumen, este TFM ha reforzado la idea de que el éxito en un proyecto de Data Analytics no solo depende de la habilidad técnica, sino también de la capacidad de adaptación, la toma de decisiones estratégicas sobre los datos y la comprensión profunda del dominio del problema.

6. Futuras mejoras

Este proyecto sienta una base sólida para una herramienta de análisis de baloncesto, pero hay numerosas vías para futuras mejoras y expansiones que podrían aumentar significativamente su valor.

- **Expansión de fuentes de datos y ligas.**
  - **Más ligas.** Integrar datos de otras ligas importantes (ACB, Eurocup, BCL) para ofrecer una visión más global. Esto requeriría adaptar los procesos ETL y los esquemas de la base de datos.
  - **Datos en tiempo real/play-by-play.** Incorporar datos en vivo o de jugada a jugada para análisis más dinámicos y detallados, permitiendo a los entrenadores reaccionar durante el partido o analizar secuencias específicas.
  - **Datos de scouting avanzado.** Integrar información cualitativa (tendencias de juego, hábitos defensivos/ofensivos específicos) que no siempre se captura en las estadísticas numéricas
- **Funcionalidades adicionales en el chat.**
  - **Análisis de tendencias.** Permitir al chatbot responder preguntas sobre tendencias a lo largo del tiempo (ej. "¿Cómo ha evolucionado el porcentaje de triples de este equipo en las últimas 5 temporadas?"), para lo cual, hemos de ampliar primero nuestra base de datos.
  - **Recomendaciones tácticas.** Basado en el análisis de datos, que el chatbot pueda sugerir ajustes tácticos o combinaciones de jugadores que ayuden al entrenador a tomar decisiones.
  - **Integración con visualizaciones.** Que el chatbot pueda generar y mostrar gráficos directamente en la conversación.
- **Dashboards y visualizaciones avanzadas.**
  - **Personalización de gráficos.** Permitir a los usuarios seleccionar las métricas exactas que desean visualizar en los dashboards y el tipo de gráfico.
  - **Mapas de tiro.** Integrar mapas de tiro detallados para jugadores y equipos, mostrando la eficiencia desde diferentes zonas de la cancha.
  - **Colores automáticos.** Usar para los gráficos los colores (principales o secundarios) del equipo que se estudiando o en el que el jugador juega.

- **Mejoras en el modelo predictivo**
    - **Más features.** Explorar la inclusión de features adicionales que puedan capturar mejor la química del equipo o el contexto del partido (ej. si los jugadores son bases, escoltas, aleros, etc., o si son tiradores, pasadores, etc.).
    - **Modelos más complejos.** Investigar arquitecturas de modelos más avanzadas (ej. redes neuronales recurrentes para secuencias de juego, o modelos gráficos para relaciones entre jugadores) si se dispone de más datos y recursos computacionales.
    - **Predicción de otras métricas.** Además del Net Rating, predecir otras métricas clave como el porcentaje de victorias o la eficiencia ofensiva/defensiva.

7. Conclusiones

Este Trabajo de Fin de Máster ha culminado en el desarrollo exitoso de una plataforma web integral para el análisis de datos avanzados en baloncesto, cumpliendo con los objetivos planteados inicialmente. La solución, construida sobre una robusta arquitectura por capas (Base de Datos, Backend con FastAPI, Frontend con Streamlit), demuestra la viabilidad y el valor de integrar diversas funcionalidades analíticas en una única herramienta accesible.

Hemos logrado construir una base de datos rica y precisa, alimentada por datos detallados de la Euroliga de HackAStat, superando las limitaciones de las APIs generales iniciales. Esta decisión estratégica fue fundamental para la profundidad de los análisis posteriores

El backend, implementado con FastAPI, actúa como el cerebro de la aplicación, orquestando funcionalidades clave como un chatbot inteligente basado en CrewAI, capaz de responder preguntas en lenguaje natural, explicar términos estadísticos y generar informes de partidos a partir de URLs. Además, el motor predictivo de quintetos, aunque un desafío complejo, ha demostrado la capacidad de estimar el rendimiento colectivo, ofreciendo una herramienta innovadora para la planificación táctica. Las funcionalidades de informes de jugadores y equipos, junto con el buscador de jugadores similares, completan un conjunto de herramientas analíticas potentes y versátiles.

El frontend, desarrollado con Streamlit, ha proporcionado una interfaz de usuario intuitiva y visualmente atractiva, haciendo que la complejidad de los datos avanzados sea accesible para entrenadores y analistas sin conocimientos técnicos profundos. Los dashboards interactivos y las comparativas visuales enriquecen significativamente la experiencia del usuario.

Las lecciones aprendidas a lo largo del proyecto, desde la importancia de la calidad de los datos hasta los desafíos de la dimensionalidad en Machine Learning y el poder de los sistemas multi-agente, han sido invaluables. Este TFM no solo es una prueba de concepto técnica, sino también un testimonio de la capacidad de transformar grandes volúmenes de datos en inteligencia accionable, un activo crucial en el baloncesto moderno.

8. Referencias bibliográficas

- CrewAI (2025). *CrewAI: Framework para la construcción de sistemas multi-agente*. Disponible en: https://www.crewai.com/
- Euroleague API (2025). *Euroleague API (boxscore_data): Librería para la extracción de datos de boxscore de la Euroliga*. Disponible en: https://github.com/giasemidis/euroleague_api
- FastAPI (2025). *FastAPI: Framework web para la construcción de APIs RESTful en Python*. Disponible en: https://fastapi.tiangolo.com/
- HackAStat (2025). *HackAStat: Fuente principal de datos avanzados de la Euroliga*. Disponible en: https://hackastat.eu/
- NumPy (2025). *NumPy: Librería fundamental para computación numérica en Python*. Disponible en: https://numpy.org/
- Pandas (2025). *Pandas: Librería para manipulación y análisis de datos en Python*. Disponible en: https://pandas.pydata.org/
- Plotly (2025). *Plotly: Librería de visualización de datos para gráficos interactivos*. Disponible en: https://plotly.com/
- PostgreSQL (2025). *PostgreSQL: Sistema de gestión de bases de datos relacionales*. Disponible en: https://www.postgresql.org/
- Psycopg (2025). *Psycopg2: Adaptador de PostgreSQL para Python*. Disponible en: https://www.psycopg.org/
- Scikit-learn (2025). *Scikit-learn: Librería de Machine Learning en Python*. Disponible en: https://scikit-learn.org/
- SportRadar (2025). *SportRadar: Proveedor inicial de datos deportivos a través de su API*. Disponible en: https://developer.sportradar.com/
- Streamlit (2025). *Streamlit: Framework de código abierto para aplicaciones web de datos en Python*. Disponible en: https://streamlit.io/

9. Índice de acrónimos

- **API**: Application Programming Interface
- **REST**: Representational State Transfer
- **AST**: Asistencias
- **AST%**: Porcentaje de Asistencias
- **AST/TO**: Ratio Asistencias a Pérdidas
- **BD**: Base de Datos
- **BLK**: Tapones
- **BLK%**: Porcentaje de Tapones
- **BLKA**: Tapones Recibidos
- **BPM**: Box Plus/Minus
- **C3**: Corner Three (Triple desde la esquina)
- **CV**: Cross-Validation (Validación Cruzada)
- **DBPM**: Defensive Box Plus/Minus
- **DF**: Faltas Recibidas
- **DNP**: Did Not Play (No jugó)
- **DR**: Rebotes Defensivos
- **DR%**: Porcentaje de Rebotes Defensivos
- **DRTG**: Defensive Rating
- **DTO%**: Porcentaje de Pérdidas de Balón en Muerto
- **eFG%**: Effective Field Goal Percentage
- **ETL**: Extract, Transform, Load
- **FGA**: Tiros de Campo Intentados
- **FG%**: Porcentaje de Tiros de Campo
- **FGM**: Tiros de Campo Anotados

- **FT**: Tiros Libres
- **FTA**: Tiros Libres Intentados
- **FT%**: Porcentaje de Tiros Libres
- **FTM**: Tiros Libres Anotados
- **GP**: Partidos Jugados
- **HGBR**: HistGradientBoostingRegressor
- **L**: Derrotas
- **L3**: Long Three (Triple Lejano)
- **LLM**: Large Language Model
- **LTO%**: Porcentaje de Pérdidas de Balón en Vivo
- **MAE**: Mean Absolute Error (Error Absoluto Medio)
- **MIN**: Minutos Jugados
- **ML**: Machine Learning
- **NetRTG**: Net Rating
- **NL2SQL**: Natural Language to SQL
- **NRTG**: Net Rating (sin distinción de ON/OFF)
- **OBPM**: Offensive Box Plus/Minus
- **OR**: Rebotes Ofensivos
- **OR%**: Porcentaje de Rebotes Ofensivos
- **ORTG**: Offensive Rating
- **PF**: Faltas Personales
- **PER**: Player Efficiency Rating
- **PG**: Point Guard (Base)
- **PF**: Power Forward (Ala-Pívot)
- **PPT**: Puntos por Tiro

- **PPP**: Puntos por Posesión
- **POSS**: Posesiones
- **PTS**: Puntos
- **R^2**: Coeficiente de Determinación
- **RMSE**: Root Mean Squared Error (Raíz del Error Cuadrático Medio)
- **SG**: Shooting Guard (Escolta)
- **SQL**: Structured Query Language
- **ST**: Robos
- **ST%**: Porcentaje de Robos
- **TFM**: Trabajo de Fin de Máster
- **TO**: Pérdidas de Balón
- **TO%**: Porcentaje de Pérdidas de Balón
- **TR**: Rebotes Totales
- **TR%**: Porcentaje de Rebotes Totales
- **TS%**: True Shooting Percentage
- **URL**: Uniform Resource Locator
- **USG%**: Porcentaje de Uso
- **UX**: User Experience (Experiencia de Usuario)
- **VAL**: Valoración
- **VORP**: Value Over Replacement Player
- **W**: Victorias
- **W%**: Porcentaje de Victorias
- **WS**: Win Shares

Anexo A. Árbol del repositorio

```
AssistAI
├── memory
├── requirements.txt
└── src
    ├── backend
    │   ├── chat
    │   │   ├── config
    │   │   │   ├── agents.yaml
    │   │   │   └── tasks.yaml
    │   │   ├── crew.py
    │   │   ├── main.py
    │   │   ├── memory.py
    │   │   └── tools
    │   │       └── boxscore.py
    │   ├── database
    │   │   ├── database.py
    │   │   └── main.py
    │   ├── main.py
    │   ├── performance_prediction
    │   │   ├── PerformancePredictor-training.ipynb
    │   │   ├── feature_columns.json
    │   │   ├── main.py
    │   │   └── performance_model.pkl
    │   ├── player_report
    │   │   ├── database.py
    │   │   └── main.py
    │   ├── search_similar
    │   │   └── main.py
    │   └── team_report
    │       ├── database.py
    │       └── main.py
    ├── config.py
    ├── etl
    │   ├── advanced
    │   │   ├── data
    │   │   │   ├── 23-24_Players.csv
    │   │   │   ├── 23-24_Teams.csv
    │   │   │   ├── 24-25_Players.csv
    │   │   │   └── 24-25_Teams.csv
    │   │   └── main.py
    │   └── basic
    │       ├── api
    │       │   └── sportradar_client.py
    │       ├── database
    │       │   └── database.py
    │       └── main.py
    └── frontend
        ├── app.py
        ├── pages
        │   ├── 1_chat.py
        │   ├── 2_player_report.py
        │   ├── 3_team_report.py
        │   ├── 4_search_similar.py
        │   └── 5_performance_prediction.py
        ├── users.yaml
        └── utils
            ├── dashboard_player.py
            ├── dashboard_team.py
            └── sidebar.py
```

Anexo B. Endpoints y parámetros

A continuación, se detallan los principales endpoints de la API RESTful implementada con FastAPI, junto con sus parámetros esperados.

Chat

- **GET /chat/response**
  - **Descripción**: Permite interactuar con el chatbot.
  - **Parámetros**:
    - `user_question` (str, Query, requerido): La pregunta del usuario en lenguaje natural.
    - `id` (str, Query, requerido): Identificador único de la sesión de conversación para mantener el contexto.
  - **Ejemplo de Uso**: `/chat/response?user_question=¿Quién fue el mejor tirador de 3 en la 2023/24?&id=12345`
- **GET /chat/get_id**
  - **Descripción**: Genera un nuevo ID de sesión para una nueva conversación con el chatbot.
  - **Parámetros**: Ninguno.
  - **Ejemplo de Uso**: `/chat/get_id`

Player Report

- **GET /player-report/report_stats**
  - **Descripción**: Obtiene estadísticas de temporada para uno o dos jugadores.
  - **Parámetros**:
    - `player1` (str, Query, requerido): Nombre del primer jugador.
    - `season1` (str, Query, requerido): Temporada del primer jugador.

        - `player2` (str, Query, opcional): Nombre del segundo jugador.
        - `season2` (str, Query, opcional): Temporada del segundo jugador.
    - **Ejemplo de Uso**:
        - Un jugador: `/player-report/report_stats?player1=Mike James&season1=2023-24`
        - Dos jugadores: `/player-report/report_stats?player1=Mike James&season1=2023-24&player2=Shane Larkin&season2=2023-24`

- **GET /player-report/season_avg_stats**
    - **Descripción**: Obtiene estadísticas promedio de todos los jugadores para una temporada específica.
    - **Parámetros**:
        - `season` (str, Query, requerido): Temporada para la que se desean las estadísticas promedio.
    - **Ejemplo de Uso**: `/player-report/season_avg_stats?season=2023-24`

Team Report

- **GET /team-report/report_stats**
    - **Descripción**: Obtiene estadísticas de temporada para uno o dos equipos.
    - **Parámetros**:
        - `team1` (str, Query, requerido): Nombre del primer equipo.
        - `season1` (str, Query, requerido): Temporada del primer equipo.
        - `team2` (str, Query, opcional): Nombre del segundo equipo.
        - `season2` (str, Query, opcional): Temporada del segundo equipo.
    - **Ejemplo de Uso**:

    - Un equipo: `/team-report/report_stats?team1=Real Madrid&season1=2023-24`
    - Dos equipos: `/team-report/report_stats?team1=Real Madrid&season1=2023-24&team2=FC Barcelona&season2=2023-24`

- **GET /team-report/season_avg_stats**
    - **Descripción**: Obtiene estadísticas promedio de todos los equipos para una temporada específica.
    - **Parámetros**:
        - `season` (str, Query, requerido): Temporada para la que se desean las estadísticas promedio.
    - **Ejemplo de Uso**: `/team-report/season_avg_stats?season=2023-24`

Search Similar

- **GET /search-similar/compare-player**
    - **Descripción**: Busca jugadores con perfiles estadísticos similares a un jugador dado.
    - **Parámetros**:
        - `player_name` (str, Query, requerido): Nombre del jugador a comparar.
        - `season_id` (str, Query, requerido): Temporada del jugador a comparar.
        - `same_role` (bool, Query, opcional, por defecto `False`): Si es `True`, compara solo con jugadores del mismo rol.
        - `weights` (Dict[str, float], Query, opcional): Un JSON string de un diccionario donde las claves son nombres de estadísticas y los valores son sus pesos (ej. `{"pts": 2.0, "ast": 0.5}`).

- **Ejemplo de Uso**: `/search-similar/compare-player?player_name=Mike James&season_id=2023-24&same_role=True&weights={"pts":2.0,"ast":0.5}`

Performance Prediction

- **GET /performance-prediction/predict**
  - **Descripción**: Predice el Net Rating de un quinteto de cinco jugadores.
  - **Parámetros**:
    - `player1_name` (str, Query, requerido): Nombre del primer jugador.
    - `season1_id` (str, Query, requerido): Temporada del primer jugador (ej. '2023-24').
    - `player2_name` (str, Query, requerido): Nombre del segundo jugador.
    - `season2_id` (str, Query, requerido): Temporada del segundo jugador.
    - `player3_name` (str, Query, requerido): Nombre del tercer jugador.
    - `season3_id` (str, Query, requerido): Temporada del tercer jugador.
    - `player4_name` (str, Query, requerido): Nombre del cuarto jugador.
    - `season4_id` (str, Query, requerido): Temporada del cuarto jugador.
    - `player5_name` (str, Query, requerido): Nombre del quinto jugador.
    - `season5_id` (str, Query, requerido): Temporada del quinto jugador.
- **Ejemplo de Uso**: `/performance-prediction/predict?player1_name=Mike James&season1_id=2023-24&player2_name=John Brown&season2_id=2023-24&player3_name=Alpha Diallo&season3_id=2023-24&player4_name=Donatas Motiejunas&season4_id=2023-24&player5_name=Donta Hall&season5_id=2023-24`

Database

- **GET /database/players**

  - **Descripción**: Obtiene una lista de todos los nombres de jugadores disponibles en la base de datos.
  - **Parámetros**: Ninguno.
  - **Ejemplo de Uso**: `/database/players`

- **GET /database/player-stats-names**
  - **Descripción**: Obtiene una lista de todos los nombres de las columnas de estadísticas de jugadores disponibles.
  - **Parámetros**: Ninguno.
  - **Ejemplo de Uso**: `/database/player-stats-names`

- **GET /database/teams**
  - **Descripción**: Obtiene una lista de todos los nombres de equipos disponibles en la base de datos.
  - **Parámetros**: Ninguno.
  - **Ejemplo de Uso**: `/database/teams`

- **GET /database/seasons**
  - **Descripción**: Obtiene una lista de todas las temporadas disponibles en la base de datos.
  - **Parámetros**: Ninguno.
  - **Ejemplo de Uso**: `/database/seasons`

Anexo C. Esquemas de las BBDD

Aquí tenemos los esquemas de ambas Bases de Datos que han sido usadas durante el desarrollo del proyecto. Comparándolas a simple vista podemos apreciar como la primera ("Basic") tiene más tablas con muchas menos columnas, lo cual, nos indica como es más general que la segunda ("Advanced"), al poder albergar datos de más ligas, mientras que la segunda es más limitada en este aspecto pero por el contrario contiene muchas más columnas en las tablas que más importancia tienen en este proyecto (aquellas que nos dan las estadísticas), lo cual nos da a entender como estos datos, a pesar de no ser tan generales, se amoldan más a nuestro caso de uso, siendo métricas de mayor valor para realizar análisis como los que hemos realizado.

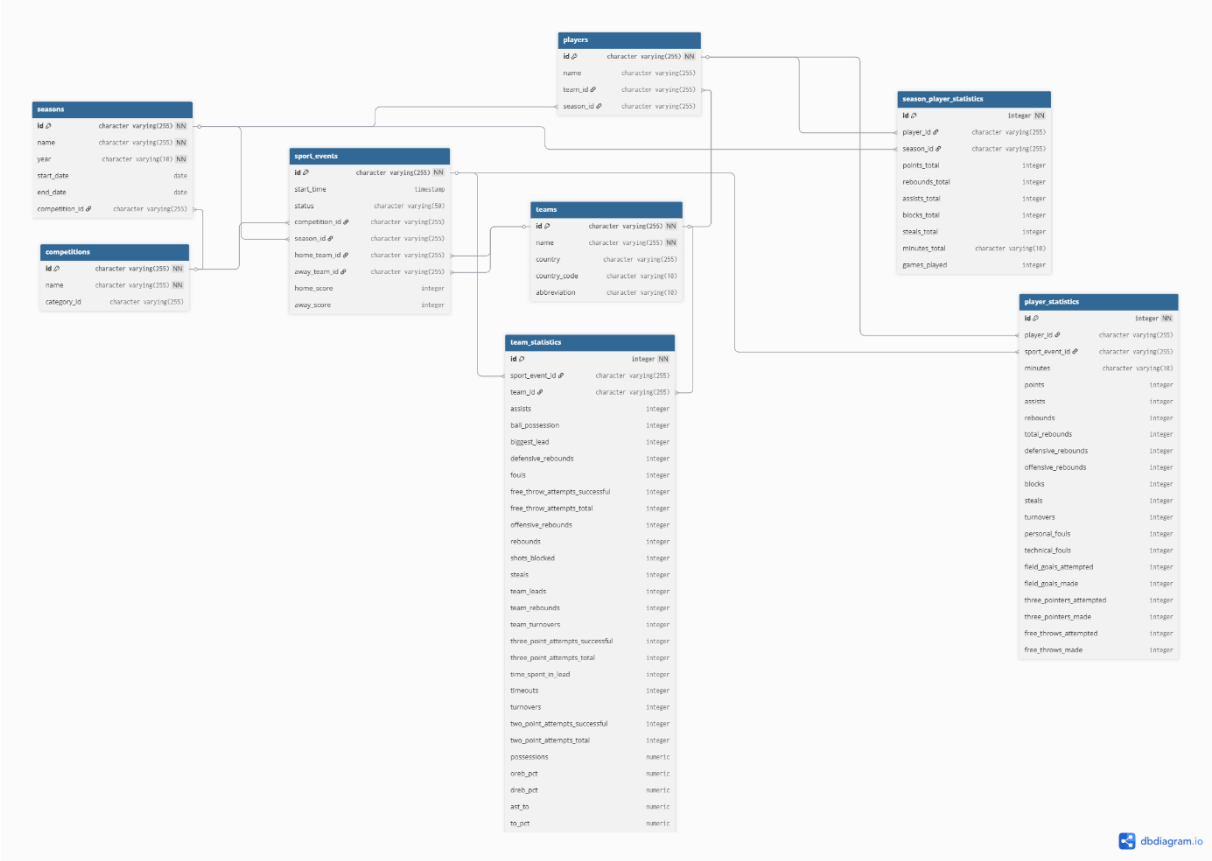

Figura 23. Esquema Base de Datos "Basic"

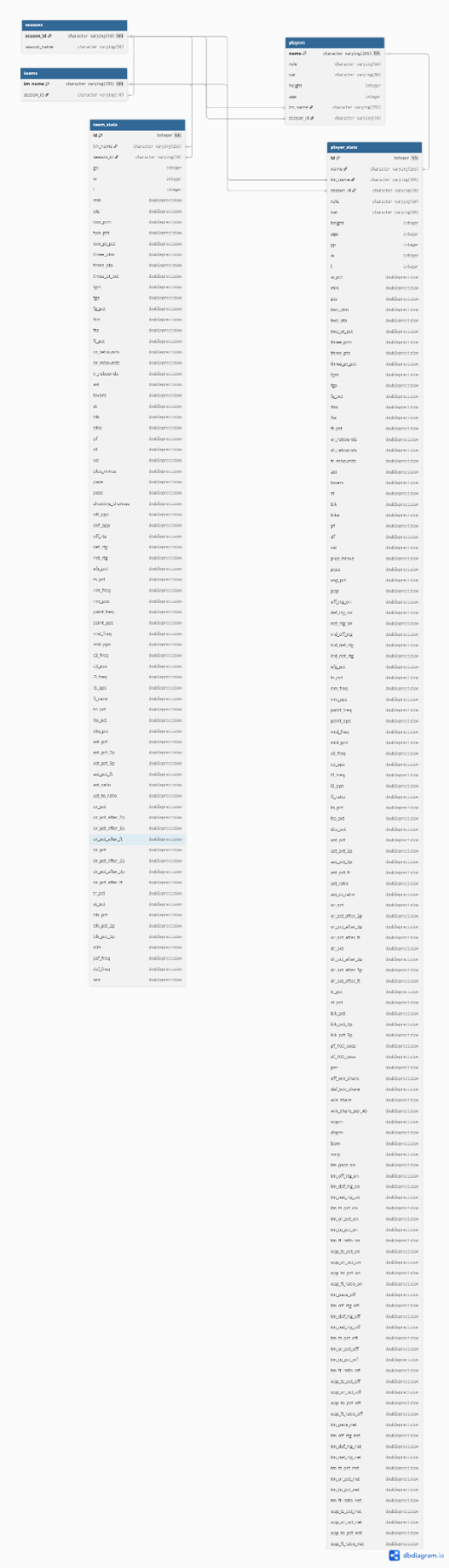

Figura 24. Esquema Base de Datos "Advanced"

Anexo D. Features del predictor

El modelo predictivo de Net Rating para quintetos utiliza un conjunto específico de características (features) que representan las estadísticas agregadas (suma y media) de los cinco jugadores en el quinteto. Estas características fueron seleccionadas tras un proceso de prueba y error, buscando el equilibrio óptimo entre la dimensionalidad y la capacidad predictiva.

El archivo `feature_columns.json` define la lista exacta de las 248 features utilizadas por el modelo `performance_model.pkl`. Cada feature se construye a partir de una estadística individual de jugador, a la que se le aplica una agregación de suma (`_sum`) o media (`_mean`) sobre los cinco jugadores del quinteto.

A continuación, se presenta la lista completa de las features utilizadas, tal como aparecen en `feature_columns.json`:


```
{
  "feature_columns": [
    "+ / -_sum",
    "+ / -_mean",
    "2PT%_sum",
    "2PT%_mean",
    "2PTA_sum",
    "2PTA_mean",
    "2PTM_sum",
    "2PTM_mean",
    "3PT%_sum",
    "3PT%_mean",
    "3PTA_sum",
    "3PTA_mean",
    "3PTM_sum",
    "3PTM_mean",
    "AGE_sum",
    "AGE_mean",
    "AST_sum",
    "AST_mean",
    "AST Ratio_sum",
    "AST Ratio_mean",
    "AST%_sum",
    "AST%_mean",
    "AST% (2P)_sum",
    "AST% (2P)_mean",
    "AST% (3P)_sum",
    "AST% (3P)_mean",
    "AST% (FT)_sum",
    "AST% (FT)_mean",
    "AST/TO_sum",
    "AST/TO_mean",
    "BLK_sum",
    "BLK_mean",
    "BLK%_sum",
    "BLK%_mean",
    "BLK% (2P)_sum",
    "BLK% (2P)_mean",
    "BLK% (3P)_sum",
    "BLK% (3P)_mean",
```

```
    "BLKA_sum",
    "BLKA_mean",
    "BPM_sum",
    "BPM_mean",
    "C3 FREQ_sum",
    "C3 FREQ_mean",
    "C3 PPS_sum",
    "C3 PPS_mean",
    "DBPM_sum",
    "DBPM_mean",
    "DEF RTG (ON)_sum",
    "DEF RTG (ON)_mean",
    "DEF WIN SHARE_sum",
    "DEF WIN SHARE_mean",
    "DF_sum",
    "DF_mean",
    "DF 100 Poss_sum",
    "DF 100 Poss_mean",
    "DR_sum",
    "DR_mean",
    "DR%_sum",
    "DR%_mean",
    "DR% (after 2P)_sum",
    "DR% (after 2P)_mean",
    "DR% (after 3P)_sum",
    "DR% (after 3P)_mean",
    "DR% (after FT)_sum",
    "DR% (after FT)_mean",
    "DTO%_sum",
    "DTO%_mean",
    "FG%_sum",
    "FG%_mean",
    "FGA_sum",
    "FGA_mean",
    "FGM_sum",
    "FGM_mean",
    "FT Ratio_sum",
    "FT Ratio_mean",
    "FT%_sum",
    "FT%_mean",
    "FTA_sum",
    "FTA_mean",
    "FTM_sum",
    "FTM_mean",
    "GP_sum",
    "GP_mean",
    "HEIGHT_sum",
    "HEIGHT_mean",
    "IND DEF RTG_sum",
    "IND DEF RTG_mean",
    "IND NET RTG_sum",
    "IND NET RTG_mean",
    "IND OFF RTG_sum",
    "IND OFF RTG_mean",
    "L_sum",
    "L_mean",
    "L3 FREQ_sum",
    "L3 FREQ_mean",
    "L3 PPS_sum",
    "L3 PPS_mean",
    "LTO%_sum",
    "LTO%_mean",
    "MID FREQ_sum",
    "MID FREQ_mean",
    "MID PPS_sum",
    "MID PPS_mean",
    "MIN_sum",
    "MIN_mean",
    "NET RTG (ON)_sum",
    "NET RTG (ON)_mean",
```

```
    "OBPM_sum",
    "OBPM_mean",
    "OFF RTG (ON)_sum",
    "OFF RTG (ON)_mean",
    "OFF WIN SHARE_sum",
    "OFF WIN SHARE_mean",
    "OPP FT Ratio (NET)_sum",
    "OPP FT Ratio (NET)_mean",
    "OPP FT Ratio (OFF)_sum",
    "OPP FT Ratio (OFF)_mean",
    "OPP FT Ratio (ON)_sum",
    "OPP FT Ratio (ON)_mean",
    "OPP OR% (NET)_sum",
    "OPP OR% (NET)_mean",
    "OPP OR% (OFF)_sum",
    "OPP OR% (OFF)_mean",
    "OPP OR% (ON)_sum",
    "OPP OR% (ON)_mean",
    "OPP TO% (NET)_sum",
    "OPP TO% (NET)_mean",
    "OPP TO% (OFF)_sum",
    "OPP TO% (OFF)_mean",
    "OPP TO% (ON)_sum",
    "OPP TO% (ON)_mean",
    "OPP TS% (NET)_sum",
    "OPP TS% (NET)_mean",
    "OPP TS% (OFF)_sum",
    "OPP TS% (OFF)_mean",
    "OPP TS% (ON)_sum",
    "OPP TS% (ON)_mean",
    "OR_sum",
    "OR_mean",
    "OR%_sum",
    "OR%_mean",
    "OR% (after 2P)_sum",
    "OR% (after 2P)_mean",
    "OR% (after 3P)_sum",
    "OR% (after 3P)_mean",
    "OR% (after FT)_sum",
    "OR% (after FT)_mean",
    "PAINT FREQ_sum",
    "PAINT FREQ_mean",
    "PAINT PPS_sum",
    "PAINT PPS_mean",
    "PER_sum",
    "PER_mean",
    "PF_sum",
    "PF_mean",
    "PF 100 Poss_sum",
    "PF 100 Poss_mean",
    "POSS_sum",
    "POSS_mean",
    "PPP_sum",
    "PPP_mean",
    "PTS_sum",
    "PTS_mean",
    "RIM FREQ_sum",
    "RIM FREQ_mean",
    "RIM PPS_sum",
    "RIM PPS_mean",
    "RNK_sum",
    "RNK_mean",
    "ST_sum",
    "ST_mean",
    "ST%_sum",
    "ST%_mean",
    "TM DEF RTG (NET)_sum",
    "TM DEF RTG (NET)_mean",
    "TM DEF RTG (OFF)_sum",
    "TM DEF RTG (OFF)_mean",
```

```
    "TM DEF RTG (ON)_sum",
    "TM DEF RTG (ON)_mean",
    "TM FT Ratio (NET)_sum",
    "TM FT Ratio (NET)_mean",
    "TM FT Ratio (OFF)_sum",
    "TM FT Ratio (OFF)_mean",
    "TM FT Ratio (ON)_sum",
    "TM FT Ratio (ON)_mean",
    "TM NET RTG (NET)_sum",
    "TM NET RTG (NET)_mean",
    "TM NET RTG (OFF)_sum",
    "TM NET RTG (OFF)_mean",
    "TM NET RTG (ON)_sum",
    "TM NET RTG (ON)_mean",
    "TM OFF RTG (NET)_sum",
    "TM OFF RTG (NET)_mean",
    "TM OFF RTG (OFF)_sum",
    "TM OFF RTG (OFF)_mean",
    "TM OFF RTG (ON)_sum",
    "TM OFF RTG (ON)_mean",
    "TM OR% (NET)_sum",
    "TM OR% (NET)_mean",
    "TM OR% (OFF)_sum",
    "TM OR% (OFF)_mean",
    "TM OR% (ON)_sum",
    "TM OR% (ON)_mean",
    "TM PACE (NET)_sum",
    "TM PACE (NET)_mean",
    "TM PACE (OFF)_sum",
    "TM PACE (OFF)_mean",
    "TM PACE (ON)_sum",
    "TM PACE (ON)_mean",
    "TM TO% (NET)_sum",
    "TM TO% (NET)_mean",
    "TM TO% (OFF)_sum",
    "TM TO% (OFF)_mean",
    "TM TO% (ON)_sum",
    "TM TO% (ON)_mean",
    "TM TS% (NET)_sum",
    "TM TS% (NET)_mean",
    "TM TS% (OFF)_sum",
    "TM TS% (OFF)_mean",
    "TM TS% (ON)_sum",
    "TM TS% (ON)_mean",
    "TO_sum",
    "TO_mean",
    "TO%_sum",
    "TO%_mean",
    "TR_sum",
    "TR_mean",
    "TR%_sum",
    "TR%_mean",
    "TS%_sum",
    "TS%_mean",
    "USG%_sum",
    "USG%_mean",
    "VAL_sum",
    "VAL_mean",
    "VORP_sum",
    "VORP_mean",
    "W_sum",
    "W_mean",
    "W%_sum",
    "W%_mean",
    "WIN SHARE_sum",
    "WIN SHARE_mean",
    "WIN Share per 40_sum",
    "WIN Share per 40_mean",
    "eFG%_sum",
    "eFG%_mean"   ],
```

```
  "best_model": "HGBR"
}
```

Cada una de estas características se mapea a una columna de la tabla `player_stats` en la base de datos. Por ejemplo, `PTS_sum` corresponde a la suma de la columna `pts` para los cinco jugadores del quinteto, y `PTS_mean` a su media. Las características que no tienen un mapeo directo en la base de datos (como `RNK`) se tratan como cero en la entrada del modelo, como se gestiona en `src/backend/performance_prediction/main.py`.