Project Description

The NBA Game Summary Generator uses statistical data from NBA games to generate human-like game summaries. The goal is to create an interactive web app where users can select a recent game, and the app will produce a recap that highlights key players, turning points, and standout performances. This project will help develop skills in data wrangling, natural language generation, and web development.

Project Steps

1. **Data Collection**
    - **Objective**: Collect game statistics and play-by-play data for recent NBA games.
    - **Sources**: Use reliable data sources to fetch comprehensive data:
        - **Official NBA API or Websites**: NBA's official API, available through NBA.com, provides game statistics, box scores, and play-by-play data.
        - **Free Public APIs**:
            - Sportsradar API (some free tiers, in-depth stats)
            - Basketball-Reference (HTML scraping for historical data)
            - DataHub.io NBA datasets (for older datasets)
    - **Tools**: You can use Python libraries like `requests` for API access and `BeautifulSoup` for web scraping.
2. **Data Preprocessing**
    - **Objective**: Structure the data for input into the OpenAI API.
    - **Steps**:
        - Extract key elements like top scorers, leading assists, game events (e.g., big plays, shifts in momentum).
        - Create structured data formats, such as JSON, capturing important stats and moments.
    - **Tools**: Use Python libraries like `pandas` to organize data into tables or JSON structures and `datetime` for any time-related transformations.
3. **OpenAI API Integration for Recap Generation**
    - **Objective**: Use the OpenAI API to generate coherent game recaps based on the structured data.
    - **Steps**:
        - **Set Up API**: Install OpenAI's Python package (`pip install openai`) and configure it with your API key.
        - **Prompt Engineering**: Develop a prompt template that uses the game stats to request a recap from the API. For example:

prompt = f"Generate a game recap for the NBA game between {team1} and {team2}. {team1} scored {score1} while {team2} scored {score2}. Key players included {player1} who scored {points1} and {player2} with {points2}. Highlight the main events and key moments."

API Call: Send a request to the OpenAI API using your structured data:

```
import openai

openai.api_key = 'your_api_key'

response = openai.Completion.create(
   engine="text-davinci-003",
   prompt=prompt,
   max_tokens=200
)
recap = response.choices[0].text.strip()
```

Tools: The OpenAI Python package is key, and you'll use prompt engineering to refine how your input data translates to natural language output.

Web Application Development

- **Objective**: Develop an interactive web interface where users can select a game and view the generated recap.
- **Recommended Tools**:
    - **Framework**: Flask or Django (Flask is lightweight and ideal for smaller applications; Django has more built-in features if you want a more robust backend).
    - **Frontend**: HTML/CSS, JavaScript, and Bootstrap for a responsive, user-friendly design.
    - **APIs**: Fetch recent games and statistics from your chosen data source within the app, allowing users to select games and view real-time recaps.
- **Steps**:
    - **Create Routes**: Set up routes in Flask or Django to handle user requests (e.g., selecting a game).
    - **Frontend Display**: Display the game data and generated recap with an intuitive design.
    - **API Integration**: Embed the OpenAI API call in your backend to process the game data and send the generated text to the frontend.

Testing and Deployment

- **Testing**: Test the app for different games and scenarios, refine prompts as needed for quality, and ensure API limits are respected.
- **Deployment**: Deploy the app on a platform like **Heroku**, **Railway**, or **Vercel** for Flask/Django apps.

Tools & Libraries Overview

- **Data Collection**: `requests`, `BeautifulSoup` (for scraping), `pandas` (data management)
- **Generative AI**: `openai` Python package for OpenAI API calls
- **Web Application**: Flask or Django for the backend; HTML, CSS, JavaScript, and Bootstrap for the frontend
- **Deployment**: Heroku, Railway, or Vercel