Project Report: FindMyPark NYC

1. Changes in Project Direction

The final project remained highly consistent with the what we proposed in the Stage 1 proposal. The core idea that creates a web application that helps NYC residents and tourists find parks based on specific needs and regions was preserved.

The primary change is that we choose not introducing environmental data. While the original proposal intended to provide information for natural habitats using the "Directory of Nature Preserves" dataset, we decided to focus the final application on athletic utility (sports, trails, and general park properties). As the "Directory of Nature Preserves" do not share the same ParkID with the previous datasets and have no detailed specification to help us understand which type of ParkID it uses, so it is less efficient to do matching with previous dataset and may reduce the accuracy.

2. Achievements and Limitations

Achievements:

We believe the application successfully achieved its goal of replacing "scattered park information" with a centralized, interactive map tool. By successfully integrating the Parks Properties, Athletic Facilities, and Parks Trails datasets, the application allows users to answer advanced queries which is difficult to do with standard navigation apps like Google Maps. The "Park Recommendation System" also successfully computes scores based on user preferences, offering a personalized experience.

Limitations:

The application failed to achieve the specific goal of providing detailed ecological data. Users looking specifically for distinct types of natural preserves will not find their targets.

3. Changes to Schema and Data Sources

Data Sources:

The source of the data remained **NYC Open Data**. However, we modified the ingestion list:

- **Retained:** Parks Properties, Athletic Facilities, Parks Trails.
- **Removed:** Directory of Nature Preserves.

Reasoning:

During the data cleaning phase, we discovered that the "Directory of Nature Preserves" dataset had significant inconsistencies in linking back to the primary "Parks Properties" dataset (IDs did not match cleanly). Furthermore, much of the "nature" data was implicitly covered by the "Parks Properties" type descriptions.

4. Changes to ER Diagram and Table Implementations

Differences:

The most significant difference between the original design and the final design is the removal of the Nature Preserves. However, in the original ER Diagram, we consider Nature Preserves as a part of `Facilities`. So there is no significant violation with the original one.

Why this is more suitable:

The final design is more suitable because it reduces sparsity in the database. The Nature Preserves dataset only covered a small subset of NYC parks (approximately 51 rows), where the Athletic Facilities and Trails has thousands of rows. Removing the table simplified our `JOIN` operations for the recommendation algorithm without significantly reducing the user experience for the majority of users.

5. Functionalities Added or Removed

Removed Functionalities:

- **Habitat Related Filtering:** We removed the ability to filter parks by specific ecological habitat types (e.g., "Wetlands") as a result of removing the Nature Preserves dataset.

Added/Refined Functionalities:

- **AI-Empowered Recommendation:** We not only implement the Preference-Based Recommendation for users to get a customized experience, but also use OpenAI API to make the users get customized experience by using a short prompt.

6. Advanced Database Programs and Application Complement

We use **Triggers** and **Stored Procedures** to keep our data accurate and make the app run faster when an update occurs. In that case, the app do not have to do heavy calculations in the web server,.

1. Keeping Ratings Accurate Automatically (Triggers)

We created triggers (`AfterReviewInsert`, `AfterReviewUpdate`, `AfterReviewDelete`) for `Review` table. Whenever a user adds, changes, or deletes a review, these triggers automatically recalculate the average rating for that park or facility.

2. Doing the Math in the Database (Stored Procedures)

We use procedures like `UpdateParkRating` to calculate averages. Instead of the app downloading every single review to calculate the score, the database does the math internally and just saves the final number.

3. Simplifying Complex Data Requests

We built procedures like `GetParkStatistics` and `GetTopRatedParksByBorough`. These act like shortcuts. Instead of the app asking for park info, then facility info, then trail info separately, it calls one procedure. The database joins all the tables, counts the facilities, and ranks the parks in one step.

7. Technical Challenges

Jieyi Zhao: The initial design of tables should not be too strict and developers should accept dynamic changes. Because as more functions are added as the development progresses, more columns may be required to support or perform optimization.

Hengrui Ren: the main challenge was user preferences were not synchronized between the front-end and the database. I fixed this by unifying all preference updates through one API, simplifying the database logic, and redesigning the front-end structure after successfully updating the database.

Yanqin Yu: One technical challenge that I found to be meaningful to mention is formatting the schemas. When I was designing the filtering function for both backend and frontend, I spent lengthy time debugging errors caused by inconsistencies of attributes for each model. This made me realize I should have taken additional care when creating the schemas during the initial steps of writing the backend structure.

Kaiyang Teng: When deploying the project to AWS, I mainly encountered three types of problems: reverse proxy configuration, frontend environment adaptation, and browser caching. First, the Nginx static file paths were still pointing to the old project directory, resulting in users accessing the old version. Second, the browser cached the old JS/CSS files, so even though the server was updated, users still saw the old version. Finally, database migration and the installation of stored procedures/triggers were also required. These problems were eventually solved by updating the Nginx configuration to point to the new path, changing the API address to use environment-based automatic detection, and adjusting the caching policy to 1 hour + must-revalidate.

8. Other Changes (Final App vs. Proposal)

- **UI Layout:** We slightly adjust the UI layout to enable better user experience.
- **Review System:** We simplified the review system slightly. The proposal suggested rating specific features (like "Trail Condition"). In the final app, we simplify the user review system to rate the park only to support a simpler recommendation system.

9. Future Work

The application could be improved by:

1. **Re-integrating Environmental Data:** We could look for a better, cleaner source of environmental data to re-introduce the "Nature Preserve" functionality we cut.
2. **Real-Time Crowd Extent:** We could allow users to report if the park is too crowded and reflect it in our map in real-time, or it is possible to integrate Google's "Popular Times" API data to adjust recommendations.
3. **Events Integration:** Incorporating an "Events" dataset from NYC Open Data to recommend parks not just based on facilities, but on upcoming activities (concerts, outdoor movies).

10. Division of Labor and Teamwork

Jieyi Zhao: Proposal writing. UML diagram design. Table design and data injection. Report writing. Some coding and debugging.
Hengrui Ren: Front-end page setting. User interaction function. UI design. AI and Map plugin import. Demo recording.
Yanqin Yu: Backend model schemas, account management, park search, park/facility/trail filtering, demo recording, proposal writing
Kaiyang Teng: AWS deployment and hosting setup, backend debugging and final testing, database setup and integration, proposal writing.