How I Started the Project:

The **Property Analysis System** project began with a focus on automating the extraction and analysis of property data, primarily from **Los Angeles City Planning (ZIMAS)** and **Tavily**. My goal was to create a system that could seamlessly scrape property data, enrich it with supplementary search results, and analyze it through AI to generate comprehensive reports.

The main challenge was ensuring that the data could flow smoothly through multiple stages: data scraping, searching, and AI analysis. Additionally, I had to focus on integrating various APIs, including **OpenRouter AI** and **Tavily**, to make the system capable of not just retrieving data, but also deriving actionable insights from it.

The Challenges I Faced:

1.Managing Multiple Agents
The project required the coordination of different agents for scraping, searching, and analyzing the data. Ensuring that these agents communicated effectively and performed in the correct order was a significant challenge. **LangGraph** was used to manage the workflow of these agents, but keeping track of each agent's performance and the flow of data was complex.

2.Integrating APIs
Integrating **OpenRouter AI** and **Tavily** posed its own set of challenges, especially when dealing with API rate limits and handling errors. I had to ensure proper handling of API keys, and also made sure the data returned by these APIs was validated before being passed to the next stage of processing.

3.Data Validation and Accuracy
One of the major issues faced during the AI analysis was ensuring the data's accuracy and consistency. I had to refine how data was being merged between the **ZIMAS** and **Tavily** results. Ensuring that missing or conflicting data was flagged was crucial to producing reliable reports.

Solutions I Implemented:

1.LangGraph for Workflow Management
By using **LangGraph**, I was able to efficiently manage the sequence of tasks performed by each agent (scraping, searching, analyzing). This made the workflow much more streamlined and ensured that data could flow seamlessly between each stage. The traceability features of **LangGraph** allowed me to monitor each step of the process.

2.Docker for Consistency
I used **Docker** to containerize the entire application, ensuring that it could run on any system without dependency issues. Docker simplified the setup and allowed me to run the project with minimal configuration. Additionally, using **Docker Compose** allowed me to manage the multiple services involved, like the **FastAPI** backend, **Streamlit** frontend, and the necessary databases.

3.API Key Management
To manage the APIs securely, I implemented **environment variables** in the .env file. This ensured that API keys for **OpenRouter** and **Tavily** were kept secure and that they could be easily configured for different environments (development, production, etc.).

4.Error Handling and Debugging
I used **LangSmith** for monitoring and debugging. This helped me trace the flow of data, understand where errors were occurring, and quickly resolve issues related to API calls and data processing.

What I Learned:

Throughout this project, I learned a lot about integrating external APIs and managing a complex data pipeline. The challenge of coordinating agents and ensuring smooth communication between them made me appreciate the value of tools like **LangGraph** for workflow orchestration. I also gained valuable experience with **Docker**, making it easier to deploy and manage the entire system in a controlled environment.

What I Would Do Differently:

Looking back, one area that could have been improved is the **error handling** when dealing with external data sources. In the future, I would implement more robust validation mechanisms to check for inconsistencies in the data at an earlier stage, possibly preventing issues further down the pipeline.