# Weekly Progress Report: Mitesh Singh

Project: MarketMind: An Open-Source Framework for AI-Driven Marketing Secondary Research

Reporting Period: July 1, 2025 – July 7, 2025

### 1. Executive Summary

This reporting period transitioned the MarketMind project from a collection of discrete components into a fully integrated, end-to-end application. The primary focus was on developing the orchestration logic to manage the AI agent crew and building an intuitive user interface to make the system's powerful capabilities accessible. Key achievements include the successful implementation of the main crew execution script, the development of a Streamlit-based web application for user interaction, and the seamless integration of the frontend and backend. The project has now reached a significant milestone: a functional prototype capable of receiving user input, conducting a comprehensive multi-agent analysis, and delivering a complete, downloadable report.

### 2. Detailed Summary of Work Completed

Following the foundational work of the previous period, this week's efforts were dedicated to weaving the individual threads—agents, tasks, and tools—into a cohesive and functional whole.

#### 2.1. Central Orchestration and Workflow Management

The core of this week's work was creating the central nervous system for the AI crew, which manages the entire research lifecycle from initiation to completion.

* **File:** main.py
* **Contribution:** I developed the primary orchestration script that serves as the entry point for any analysis. This script is responsible for:
  + **Initialization:** It instantiates the MarketResearchAgents and MarketResearchTasks classes.
  + **Agent & Task Assembly:** It calls the specific methods to create instances of all five agents and their corresponding tasks.
  + **Dependency Injection:** It correctly injects the task dependencies, ensuring that the Critique and Synthesis tasks receive the outputs from the preceding analysis tasks as their context.
  + **Crew Formation:** It assembles the agents and tasks into a Crew object, configuring it to run in a Process.sequential manner. This enforces the logical workflow where research planning is followed by analysis, critique, and final synthesis.
  + **Execution:** It triggers the entire process using crew.kickoff(), passing the user-defined inputs. Upon completion, it writes the raw output from the final Synthesis task into a markdown file (final\_market\_analysis\_report.md), providing a persistent record of the analysis.

#### 2.2. User Interface (UI) Development for Accessibility

A key goal of MarketMind is to democratize access to marketing AI. To this end, I developed a simple yet effective web interface.

* **File:** app.py
* **Contribution:** I built a complete user-facing application using Streamlit. This UI abstracts away all the backend complexity, presenting the user with a clean and intuitive interface. The key features include:
  + **Structured Input Fields:** Using st.text\_input, st.text\_area, and st.selectbox, I created a sidebar form that prompts the user for all necessary information: Product Name, Product Description, Target Industry, Geography, and Company Scale.
  + **Form-Based Submission:** The inputs are wrapped in an st.form to ensure that the analysis is only triggered when the user explicitly clicks the "Start Comprehensive Analysis" button, preventing accidental executions.

#### 2.3. Frontend-to-Backend Integration

Connecting the user-friendly UI to the powerful backend crew was a critical integration task that required a robust and secure solution.

* **File:** app.py
* **Contribution:** I engineered the communication bridge between the Streamlit frontend and the crewai backend.
  + **Process Execution:** I utilized Python's subprocess module to run the main.py script as a separate process. This is a robust method that isolates the UI from the potentially long-running analysis task.
  + **Secure Data Transfer:** User inputs from the Streamlit form are passed to the backend script as environment variables (os.environ). This is a secure and standard way to pass configuration data to a child process without exposing it directly as command-line arguments. The main.py script then retrieves these variables using os.getenv().

#### 2.4. Report Presentation and Delivery

The final piece of the user journey is the delivery of the research findings. I implemented a system to present the final report directly within the application.

* **File:** app.py
* **Contribution:** Once the backend subprocess completes, the Streamlit application automatically:
  + **Reads the Report:** It opens and reads the contents of the final\_market\_analysis\_report.md file.
  + **Displays the Content:** It uses st.markdown to render the report's content within an st.expander, allowing the user to view the full analysis directly on the web page.
  + **Enables Download:** It provides an st.download\_button, which allows the user to save the complete markdown report to their local machine for offline use, sharing, or further editing.

### 3. Challenges Encountered & Solutions

* **Challenge:** The AI crew's analysis is computationally intensive and can take several minutes to complete. During this time, a static UI would appear frozen, leading to a poor user experience and potential user abandonment. It was critical to provide clear feedback that the system was working.
* **Solution:** I addressed this by implementing an asynchronous user feedback mechanism in the Streamlit app.
  + By wrapping the subprocess.run() call within an st.spinner context manager, the UI displays an animated loading indicator and a clear message ("Your AI crew is conducting the market research..."). This immediately reassures the user that their request has been received and is being processed.
  + Furthermore, I configured the subprocess call with capture\_output=True to prevent the verbose backend logs from flooding the clean UI. The logs are suppressed during normal operation but are captured and displayed if an error occurs (process.returncode != 0), which is essential for debugging. This approach provides the best of both worlds: a clean UI for the user and detailed error information for the developer.

### 4. Goals for the Next Reporting Period (July 8 - July 15, 2025)

With a functional end-to-end prototype now complete, the focus will shift from core feature development to quality assurance, refinement, and documentation.

1. **Rigorous Testing & Quality Assurance:** Conduct extensive testing using a wide variety of inputs (different products, industries, and scales) to evaluate the quality, coherence, and robustness of the AI-generated reports.
2. **Prompt Engineering and Refinement:** Based on the test results, I will iterate on the agent prompts and task descriptions. The goal is to fine-tune the instructions to elicit more insightful, accurate, and contextually relevant analysis from the LLMs.
3. **Code Refactoring and Documentation:** I will perform a thorough code review, refactoring where necessary to improve clarity, maintainability, and performance. I will also add comprehensive inline comments and docstrings to all functions and classes, which is critical for a successful open-source project.
4. **Develop Project README:** I will begin drafting the main README.md for the project repository. This document will serve as the primary guide for new users and contributors, detailing the project's architecture, setup instructions, and usage examples.

### 5. Hours Contributed During This Period

* **Total:** 20 hours