Lessons Learned: Kaggle 5 Days of AI (Google Agents)

1. Agent Architecture (Day 1)

- **Pattern**: Use a clear separation between the **Reasoning Engine** (LLM) and the **Runtime** (Code/Tools).
- **Class Structure**:
  - Define an `Agent` class that holds `model`, `tools`, and `memory`.
  - Use `vertexai.preview.reasoning_engines` (or similar SDK) to wrap the agent for deployment.
  - **Configurability**: The model name (e.g., `gemini-1.5-pro`) should be passed as an argument to the Agent constructor, allowing easy switching.

2. Tools (Day 2)

- **Definition**: Tools should be Python functions with clear docstrings (Google-style).
- **Registration**: Use `ReasoningEngine.register_tools()` or pass `tools=[func1, func2]` to the Gemini model.
- **Best Practice**:
  - Type hints are mandatory.
  - Docstrings must describe *when* to use the tool and *what* the arguments are.
  - Return structured data (JSON/Dict) rather than long text strings where possible.

3. Memory & Sessions (Day 3)

- **Session State**:
  - Do not store state in the Agent class instance if deploying to serverless (Cloud Run).
  - Use an external store (Firestore/Redis) keyed by `session_id`.
- **Context Management**:
  - Use a `History` object that is loaded at the start of each turn and saved at the end.
  - Implement "Context Compaction" (summarizing old turns) if the history gets too long.

4. Observability (Day 4)

- **Logging**:
  - Log every "Thought", "Action", and "Observation".
  - Use structured logging (JSON) to make it queryable in Cloud Logging.
- **Evaluation**:
  - Use `ragas` or Vertex AI Eval to score "Faithfulness" and "Relevance".

5. Agent-to-Agent (Day 5)

- **Orchestration**:
  - Use a "Router" or "Orchestrator" pattern where the main agent's only tools are "delegate_to_researcher" and "delegate_to_analyst".
  - **Protocol**: Pass a clear `task_description` and `context` to the sub-agent. The sub-agent returns a `final_answer`.

Implementation Checklist

- [ ] **Config**: Create `config.py` to load `MODEL_NAME` from env or default.
- [ ] **Base Agent**: Create a `BaseAgent` class that handles tool registration and memory loading.
- [ ] **Tools**: Ensure all tools (PubMed, CT) have perfect docstrings.
- [ ] **Orchestrator**: Implement the main loop that delegates to sub-agents.