This version includes notes where Cursor or AI-assisted coding can help accelerate your progress and improve code quality — all while keeping the focus on learning, showcasing your product skills, and delivering a real, demo-ready MVP.

**AI-Driven Fraud Detection Dashboard – MVP Project Plan (With Python + AI Coding Assistant Support)**

**Overview**

**Project Goal:**  
Build an interactive dashboard that simulates fraud detection by analyzing transaction data. The goal is to create a compelling portfolio project demonstrating your understanding of fraud, risk management, data-driven product strategy, and how AI tools can automate and enhance decision-making.

As agent – recommend the best agentic workflow for this demo (e.g. augmented LLM using memory and tool, evaluator, other??) – my thinking is an augmented LLM model is sufficient but please make alternative suggestions as helpful

**1. Target Audience**

* **Hiring Managers:** Product roles in AI, FinTech, and RiskOps.
* **Technical Teams:** Show you understand data pipelines, ML basics, and system architecture.
* **PM Community:** Demonstrates how a PM can think like a builder—even without deep engineering skills.

**2. Core Features (MVP Scope) – use updated section #2 further below**

| **Feature** | **Description** | **Implementation Support** |
| --- | --- | --- |
| **Data Ingestion Tool** | Upload and read a CSV file containing mock transactions. | Use Python’s pandas and streamlit.file\_uploader. Ask Cursor for "Python CSV upload with pandas". |
| **Anomaly Detection Tool** | Use simple ML (e.g., Z-score or Isolation Forest) to detect suspicious behavior. | Ask Cursor for "Anomaly detection using scikit-learn" and tweak examples. |
| **Alerting Tool** | Generate alert cards for transactions flagged as suspicious. Include metadata like amount, time, location. | Ask Cursor for a “Streamlit alert card or transaction card UI”. |
| **Memory (Learning Feedback Loop)** | Allow a user to mark a transaction as false/true positive and save result locally. | Use a local CSV or lightweight JSON file. Ask Cursor how to “save feedback to file in Python”. [update- use JSON and recommend key/value pairs)] |
| **Visual Dashboard** | Display charts showing volume of alerts, types of transactions, and anomaly scores over time. | Use plotly, seaborn, or matplotlib. Cursor can generate the visuals for you. |

**3. Recommended Tech Stack (Beginner + AI Assistant Friendly) use updated section #3 further below**

| **Component** | **Technology** | **Why** |
| --- | --- | --- |
| **Language** | Python | Strong community, simple syntax, and AI code assistant support. |
| **IDE** | Cursor | Auto-suggests clean Python code and handles bugs and refactors. |
| **Notebook** | Jupyter or Google Colab | Great for testing ML logic before dashboard integration. I’m not familiar w/Colab – use Jupyter |
| **Dashboard UI** | Streamlit | Fastest way to build a Python-based dashboard. |
| **ML Library** | Scikit-learn | Beginner-friendly and well-documented. |
| **Data Format** | CSV | Easy to mock, edit, and store local feedback (e.g., labels). |

**4. Project Folder Structure – use updated section #4 below**

Fraud\_detection\_dashboard/

│

├── data/

│ └── sample\_transactions.csv # Mock transaction dataset

│

├── notebooks/

│ └── detect\_anomalies.ipynb # ML logic and data exploration

│

├── app/

│ ├── dashboard.py # Streamlit app

│ ├── ingest.py # CSV upload & parsing logic

│ ├── detect.py # Anomaly detection logic

│ ├── alert.py # Alert creation logic

│ └── memory.py # Saves user feedback

│

├── assets/

│ └── screenshots/ # Portfolio visuals

│

└── README.md # Overview & instructions

[update to include .venv so I can include environmental variables and also ‘ignore when uploading to git]

[update to include a configurable profile file – JSON would be best -where a user (me in this case) can provide the name of a company -e.g., a streaming service like Netflix or some other recuring charge like for chewy.com along with a threshold for each – for example “frequency:monthly, name:chewy, threshold:$25.00 – the thinking is ‘detect.py would check here as an initial step before calling AI or py module]

**5. Milestones & Timeline (6-Week Plan)**

| **Week** | **Tasks** | **AI Assistant Prompts (Cursor)** |
| --- | --- | --- |
| **Week 1** | Define MVP scope, gather mock data (CSV), sketch UI. | “Create fake bank transactions CSV using Python” |
| **Week 2** | Set up data ingestion using pandas + Streamlit. | “Streamlit file uploader with pandas parser” |
| **Week 3** | Implement anomaly detection (Z-score or Isolation Forest). | “Detect outliers using Isolation Forest in Python” |
| **Week 4** | Build alert cards and feedback capture system. | “Streamlit card for flagged transactions + button to mark false positive” |
| **Week 5** | Build visualizations of alert frequency, amount over time, category trends. | “Plotly time series chart of transactions” |
| **Week 6** | Finalize design, test flows, take screenshots, and prepare a walkthrough. | “Save user interaction (feedback) to local file” |

**6. Portfolio Positioning Tips**

Include this in your portfolio with:

* GitHub repo (link to working code, README with setup instructions)
* Short Loom/OBS video showing the full workflow
* Slides or one-pager summarizing:
  + Problem: Fraud detection is costly and manual
  + Solution: Automated anomaly detection dashboard
  + Your role: Designed the flow, modeled data, iterated UX, used AI tools to build it

**7. Example Demo Flows - use updated section #7 further below**

1. User uploads CSV.
2. Dashboard parses and displays transaction summary.
3. Outliers are flagged in red with context.
4. User can click to mark alerts as accurate or false positives.
5. Feedback updates local memory file (JSON or CSV).
6. Visuals update accordingly.

**8. Helpful AI Assistant Prompts (for Cursor or ChatGPT)**

* “Build a Streamlit dashboard that uploads a CSV and displays a table”
* “Anomaly detection using scikit-learn and Isolation Forest”
* “Create a transaction alert card in Streamlit”
* “Track user feedback and store it in a CSV”
* “Plot transaction anomalies over time using Plotly”
* “Add simple stateful memory to a Python Streamlit app”

You can also paste partial code and ask for:

* **Bug fixes**
* **Code explanations**
* **Refactors for readability or reusability**

**9. Future Enhancements**

* Add a login simulation (Streamlit Auth or Firebase)
* Connect to a SQLite DB for persistent memory
* Add classification model (fraud vs. non-fraud)
* Generate natural language explanations for alerts

**10. Final Tips**

* Let Cursor handle **repetitive logic, visualization boilerplate, and ML syntax**.
* Use your **PM strengths** to:
  + Define smart rules for what counts as “suspicious”
  + Prioritize UX clarity and alerting hierarchy
  + Document user stories and success metrics
* Show your project as a bridge between **domain expertise** and **technical execution**.

Would you like this delivered as a **Word (.docx)** file or a **Google Docs version**? I can also help build out the README.md or slide deck version if you’re preparing this for interviews or networking.

[Updated section 2]

| **Feature** | **Description** | **Implementation Support** |
| --- | --- | --- |
| **Data Ingestion Tool** | Upload and read a CSV file containing mock transactions. | Use Python’s pandas and streamlit.file\_uploader. Ask Cursor for “Python CSV upload with pandas.” |
| **Anomaly Detection Tool** | Use simple ML (e.g., Z-score or Isolation Forest) to detect suspicious behavior. | Ask Cursor for “Anomaly detection using scikit-learn” and tweak examples. |
| **Alerting Tool** | Generate alert cards for transactions flagged as suspicious. Include metadata like amount, time, location. | Ask Cursor for a “Streamlit alert card or transaction card UI.” |
| **Memory (Learning Feedback Loop)** | Allow a user to mark a transaction as false/true positive and save result locally. | Use a lightweight **JSON file** with key/value pairs. Ask Cursor how to “save feedback to JSON in Python.” |
| **Visual Dashboard** | Display charts showing volume of alerts, types of transactions, and anomaly scores over time. | Use Plotly, Seaborn, or Matplotlib. Cursor can generate the visuals for you. |
| **Configurable Profile Settings** | Allow user to configure known recurring charges (e.g., Netflix, Chewy) and set thresholds for anomaly detection. | Store settings in a separate JSON file. Ask Cursor for “Read and use config from JSON in Python.” |
| **Environmental Management** | Use a .venv folder for Python virtual environment and ignore it in Git. | Use .gitignore to avoid uploading environment dependencies. Cursor can help automate environment setup. |

[Updated section 3]

| **Component** | **Technology** | **Why** |
| --- | --- | --- |
| **Language** | Python | Strong community, simple syntax, and AI code assistant support. |
| **IDE** | Cursor | Auto-suggests clean Python code and handles bugs and refactors. |
| **Notebook** | Jupyter | Great for testing ML logic before dashboard integration. |
| **Dashboard UI** | Streamlit | Fastest way to build a Python-based dashboard. |
| **ML Library** | Scikit-learn | Beginner-friendly and well-documented. |
| **Data Format** | CSV, JSON | CSV for raw transactions, JSON for user feedback & config. |

[Updated section 4]

Fraud\_detection\_dashboard/

│

├── .venv/ # Python virtual environment (excluded via .gitignore)

│

├── config/

│ └── recurring\_config.json # JSON config for recurring charges (e.g., Netflix, Chewy)

│

├── data/

│ └── sample\_transactions.csv # Mock transaction dataset

│

├── notebooks/

│ └── detect\_anomalies.ipynb # ML logic and data exploration

│

├── app/

│ ├── dashboard.py # Streamlit app

│ ├── ingest.py # CSV upload & parsing logic

│ ├── detect.py # Anomaly detection logic (uses config for known charges)

│ ├── alert.py # Alert creation logic

│ └── memory.py # Saves user feedback to JSON

│

├── assets/

│ └── screenshots/ # Portfolio visuals

│

├── .gitignore # Exclude .venv, local files

└── README.md # Overview & setup instructions

[Updated section 7]

**7. Example Demo Flows**

1. User uploads CSV of transactions
2. Dashboard parses and shows transaction summary
3. Detect module checks config file for known recurring charges (e.g., Netflix, Chewy)
4. Outliers are flagged and displayed in alert cards
5. User marks alerts as accurate or false positives
6. Feedback is saved to a JSON file
7. Visualizations update in real time based on feedback