# Project Implementation Report

## Google Issue Tracker AI Triage

Date: 2026-01-21

## Table of Contents:

# 1. Requirements Analysis

A straightforward comparison of your requirements versus what is currently implemented.

| Requirement Goal | Status | Implementation Notes |
|---|---|---|
| Step 1: Data Collection | PARTIAL | We extract ID, Title, Description, but not full 'Labels', 'Creation D |
| - Read Issue Tracker ID/Title | DONE | Reads from input CSV. |
| - Read Status/Labels/Dates | PARTIAL | Dependent on CSV columns. Scraper fetches descriptions. |
| Step 2: Category 1 Classification | DONE | Logic implemented in `detect_pixel_model` and `categorize_issu |
| - Primary Categories (Pixel X, Network) | DONE | Basic keyword matching is implemented. |
| - AI/Semantic Understanding | BASIC | Currently uses Keyword/Regex matching, not LLM-based under |
| Step 3: Visualization | DONE | Bar charts and Summary tables are generated automatically. |
| - Count & Visualize | DONE | Matplotlib/Seaborn charts generated. |

**Summary:** We have a functional MVP. It successfully reads, scrapes, categorizes using logic rules, and visualizes the data. To truly meet the 'AI' requirement (semantic understanding), we would need to integrate an LLM API (like GPT/Gemini) instead of just Keyword matching.

# 2. How to Run This Project

Follow these steps to execute the project on your local machine.

## Prerequisites:

- Python 3.8 or higher installed.
- An active internet connection.
- The input CSV file placed in `input/` folder.

## Step 1: Install Dependencies

Open your terminal in the project folder and run:

```
pip install -r requirements.txt
```

## Step 2: Run the Scraper

Execute the main script:

```
python main.py
```

*Note: The script will automatically install missing dependencies if Step 1 was skipped.*

## Step 3: Check Results

Navigate to the `output/` directory to see:

- **cleaned_data.csv:** The fully processed dataset.
- **summary.csv:** Classification counts.
- **charts/**: Visual graphs of the data.

# 3. Technical Implementation Details

The solution is a Python-based pipeline designed for speed and reliability.

## Core Logic:

- **Data Ingestion:** Reads CSV files using Pandas, handling various encoding formats (utf-8, latin-1) automatically.
- **Multithreaded Scraping:** Uses `concurrent.futures` to scrape 20 pages in parallel. This was crucial for performance. It extracts descriptions from HTML using BeautifulSoup.
- **Categorization Engine:** A rule-based classifier assigns categories.
  *- e.g., If description contains 'Pixel 9', assign 'Pixel 9 Series'.*
  *- e.g., If description contains 'wifi', assign 'Network Issues'.*

# 4. Challenges Faced & Solutions

## Challenge 1: SSL Certificate Verification Errors

**Problem:** The script initially failed with `[SSL: CERTIFICATE_VERIFY_FAILED]`. This often happens in corporate network environments with proxies.
**Solution:** We modified the `requests.get()` call to include `verify=False` and suppressed the resulting warnings using `urllib3`. This allows the scraper to function correctly in your environment.

## Challenge 2: Performance (Slow Scraping)

**Problem:** Scraping ~800 issues one-by-one with Selenium or sequential logic would take 20-30 minutes.
**Solution:** We switched to a **multithreaded** approach using Python's `ThreadPoolExecutor`. This reduced the execution time to under 1 minute by fetching 20 pages simultaneously.

## Challenge 3: Robustness

**Problem:** The initial Selenium attempt was unstable and timed out.
**Solution:** We reverted to `requests` + `BeautifulSoup` which is lighter and faster for static content. We kept the code modular so scraping logic is separate from analysis logic.