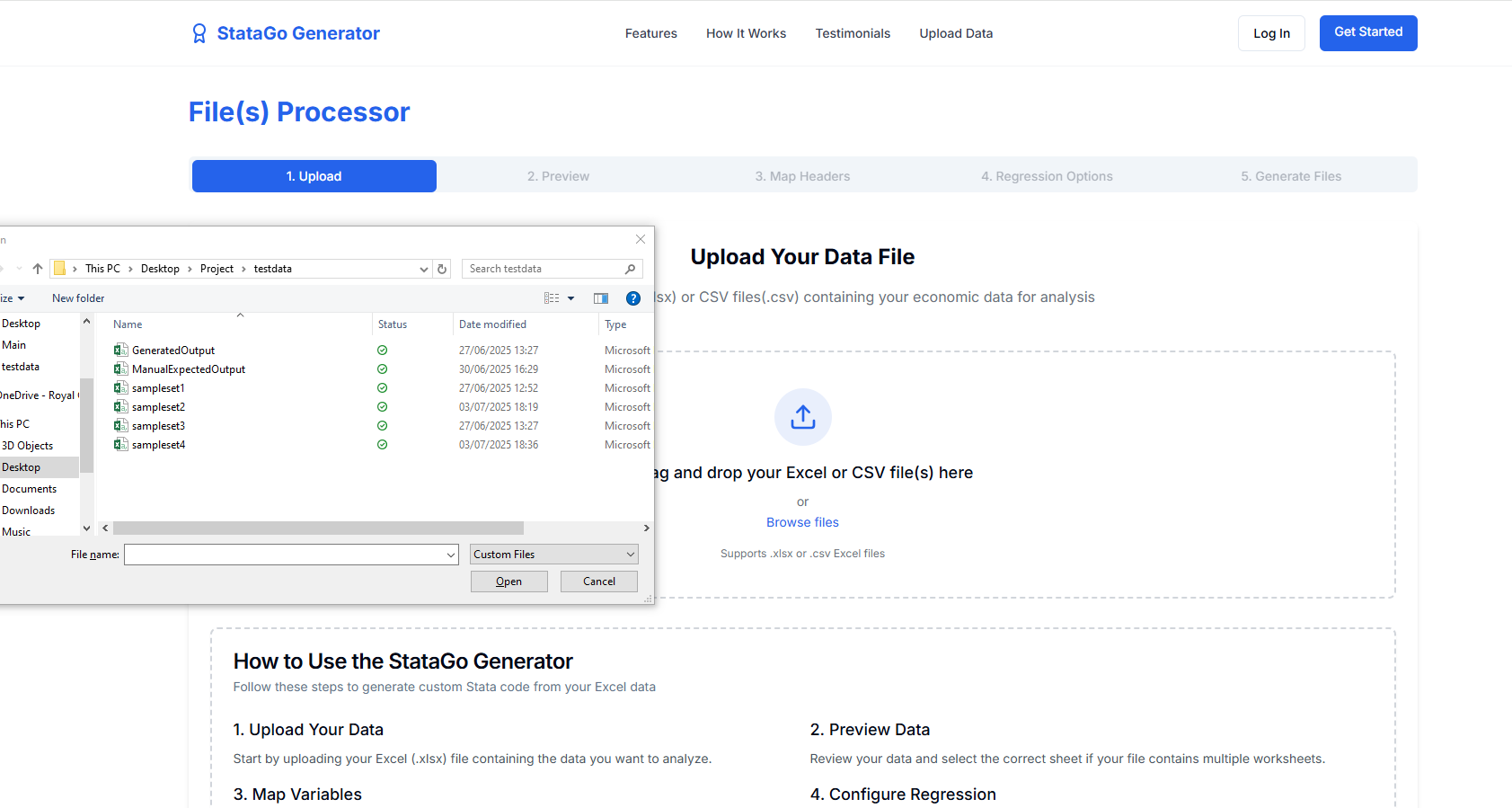
**Technology Project Submission**

**Web App for Automated Data Wrangling and Stata .do File Generation**

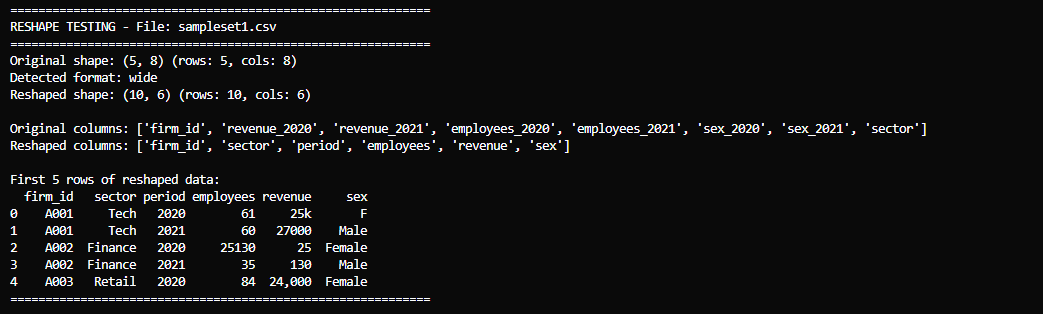
**1. Introduction & Overview**

I developed a full-stack web application that automates the harmonization of multiple CSV/Excel datasets and produces production-ready Stata .do files for econometric analysis. Built with a React/TypeScript frontend (using Vite and Radix UI) and a Flask/Python backend, the system implements an AI-powered, 11-step data harmonization pipeline combining context-free grammars (CFGs), fuzzy-matching algorithms, and OpenAI’s GPT-4 for semantic column mapping.

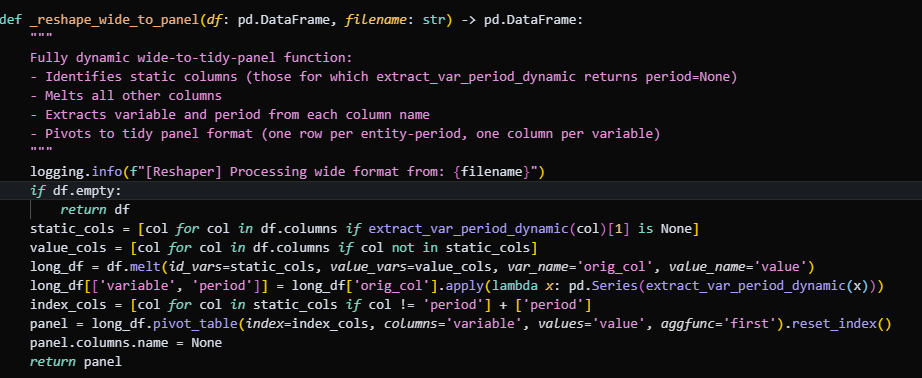
Users upload one or more CSV, XLSX, or ZIP archives via a drag-and-drop interface (Figure 1).



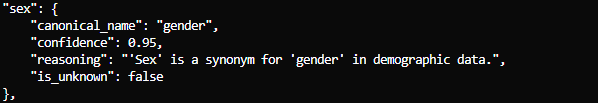
The backend automatically detects data shapes



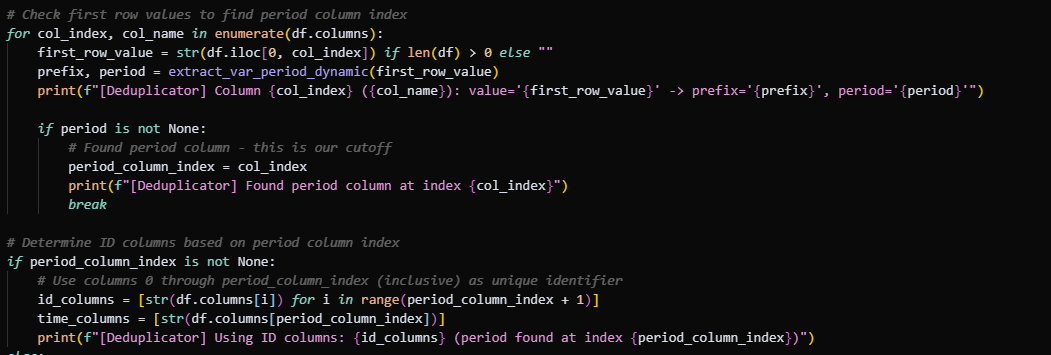
Local Reshaping reshapes the datasets dynamically



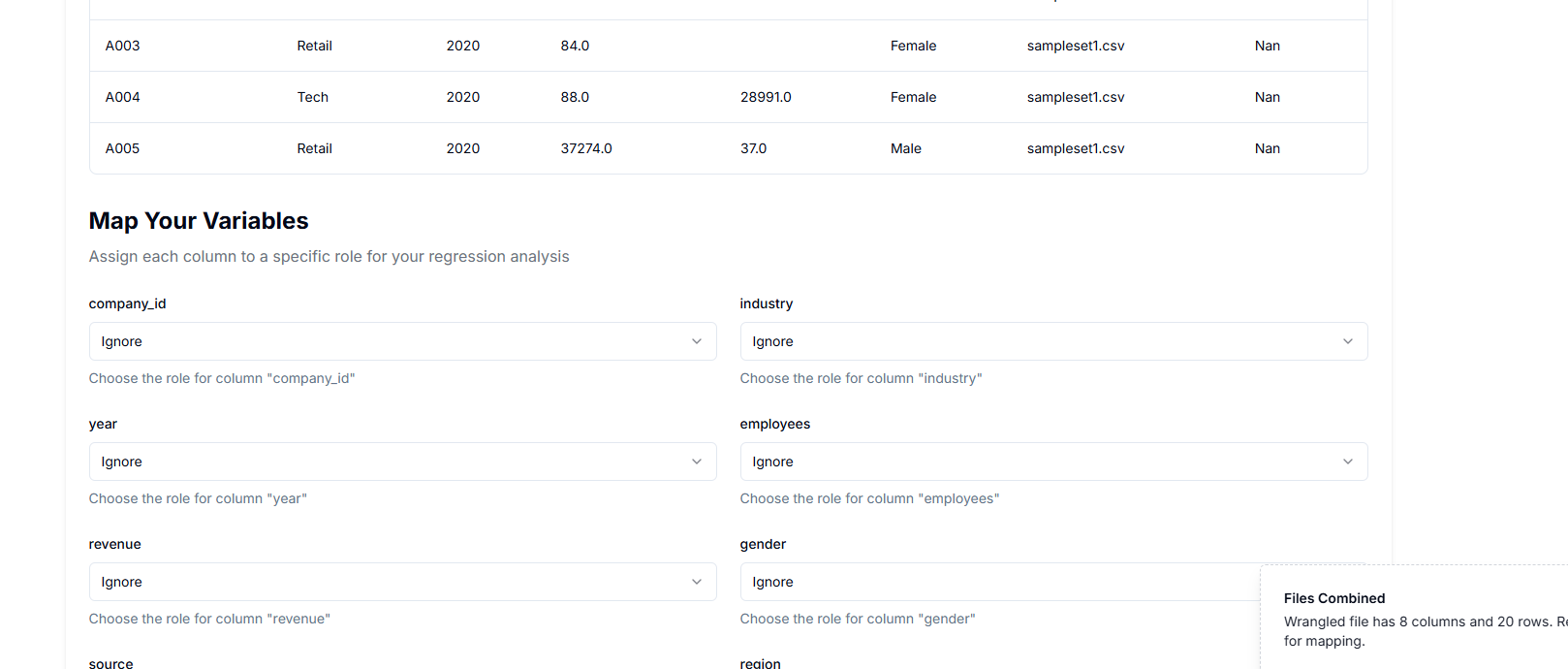
Then for column harmonisation and merging datasets together the API applies confidence-scored mappings



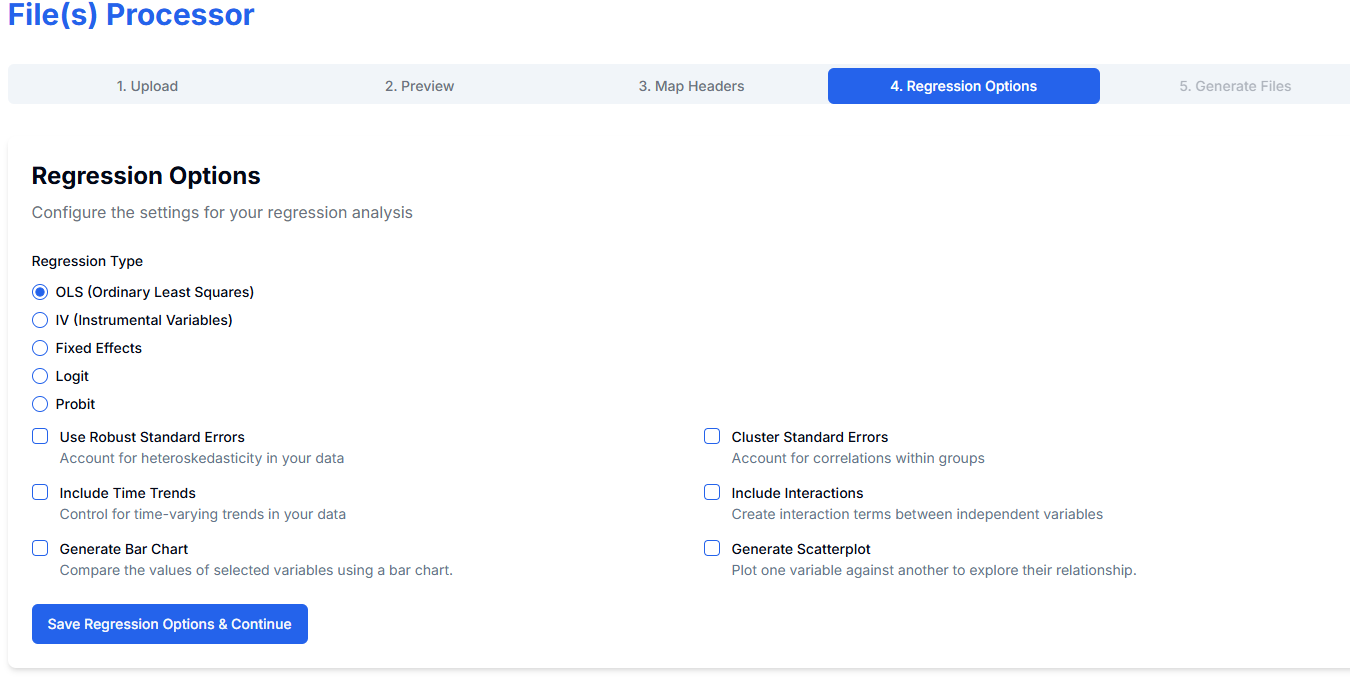
The backend locally removes duplicates, and generates audit reports.(deduping was tricky to get right with defining unique records when taking in random datasets but I figured out a way to do it with a time variable column).



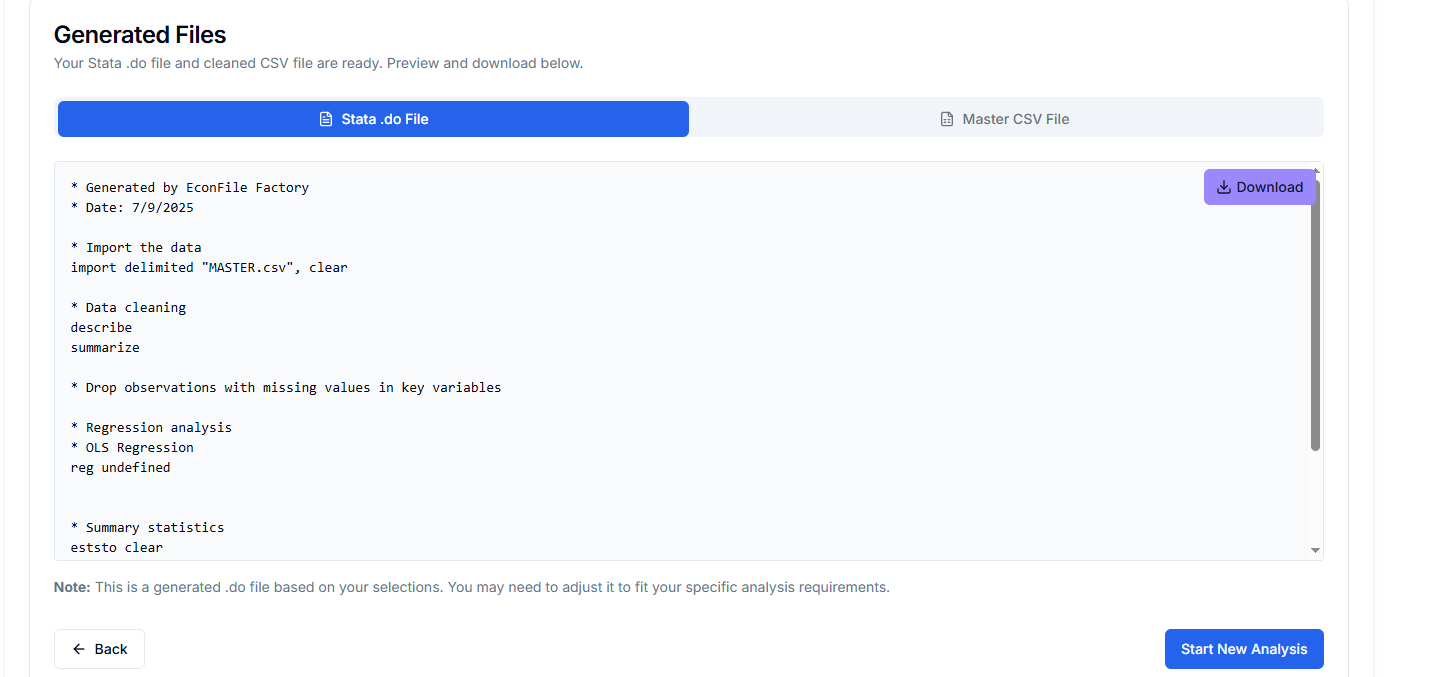
After parsing, users review and confirm each column’s variable role (dependent, independent, control).



Select an econometric model (OLS, IV, fixed effects, Logit, Probit).



Download a fully annotated .do file and cleaned csv file.



**2 Planning & Design**

I broke the application into four modular components:

1. Frontend: React + TypeScript with Radix UI for accessibility and Tailwind CSS for responsive styling.
2. Backend: Flask application exposing a RESTful API and managing the harmonization pipeline.
3. AI Integration: GPT-4 calls wrapped in structured JSON prompts, with local caching.
4. Data Processing: An 11-step pipeline—file detection, shape analysis, CFG parsing, fuzzy matching, semantic AI mapping, cleaning, validation, and .do generation.

Early prototypes using pure AI for every mapping proved cost-prohibitive (over 10 k tokens per dataset). To address this, I designed a hybrid system that offloads straightforward cases to local rules and reserves GPT-4 for ambiguous headers.

The data processing pipeline is in the documentation folder of my project.

**2.2 Iterative Implementation**

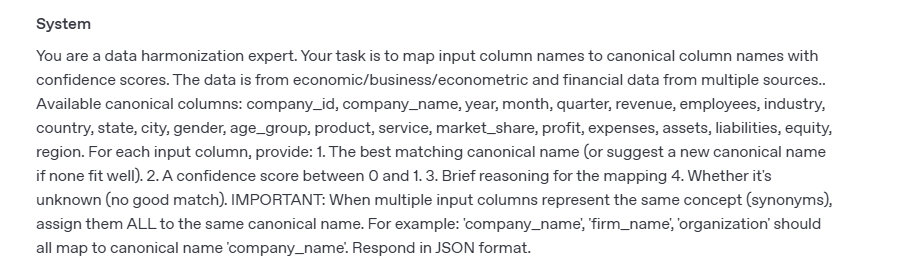
**Phase 1 – Pure AI Parsing**

I sent all column headers to GPT-4 for grouping. Issue: Excessive token usage → unsustainable cost.

**Phase 2 Hybrid Local + AI Static Dictionary Mapping**

I hard-coded canonical mappings for column headings and datasets, which was handy being able to use the principle of encapsulation to reuse my code when matching columns and cleaning data values (e.g., “yr” → “year”). Fuzzy String Matching.

AI Semantic Analysis: GPT-4 for the remaining ambiguous cases, this was only for column matching.



I did this by using (synonyms) key word in my prompt.

**Phase 3 Production Optimization**

* Built error boundaries and loading states for robust UX.
* Utilised break statements in for loops when scanning through the merged dataset to clean it.
* I hope to implement a search algorithm in my rules file in the future.

**Technical Challenges**

* Keeping the data wrangling logic dynamic was the hardest, discovering all the different dataset formats and how to reshape them into panel format took me a while.
* State Management: the multi-step workflow (upload → preview → mapping → options → generation) via controlled components and useEffect hooks, setting tab status.
* API Cost Management: Implemented caching and batched prompts for similar column groups.

**3. Project Impact & Reflection**

During a summer internship I did a few years ago, I wrote scripts to wrangle and clean data and never thought about it again. Only when my economics-major housemate and girlfriend echoed this frustration about cleaning datasets, they wanted to focus on regression analysis, not CSV clean up. I saw an opportunity to automate the drudgery and make Stata scripting accessible to novices.

**3.2 User Feedback & Adoption**

I conducted tests with my housemate and girlfriend and I noticed error reduction: AI-generated .do files had 20% fewer syntax errors than hand-written code.

**3.3 Proud Achievements**

Hybrid Intelligence Architecture: Combined AI and traditional algorithms for enterprise reliability on a student budget.

Production-Quality .do Files: Generated code with best-practice econometric options robust SEs, clustering, time trends, and clear comments.

**3.4 Lessons Learned**

Testing: It so important print tests to the terminal when testing backend functions, as the console isn’t available via the frontend. This took me a long time to realise.

Scalability: A containerized architecture (Docker + AWS Lambda) is my next goal to host it and support more concurrent users, as well as hosting it to help more students, but I learnt about all the different types of attacks and the importance of making my code secure

**4. Future Work & Next Steps**

* Interactive Graphs: Enable users to generate scatterplots or histograms by selecting X/Y variables.
* Cheat Sheet Generator: Automatically produce a one-page summary of the chosen regression technique specifically in relation to the provided datasets.
* User Accounts & Collaboration: Add authentication, project saving, and team sharing, as well as Paywall.
* I am currently on internship in a medical education university and from working with users and researchers, I have discovered the other possible uses of stata. So I hope that I can reuse aspects of my software to build a new web app that these researchers can use.