School of Computing and Mathematical Sciences

CO7201 Individual Project

Final Report

Exploring HESA Data with Large Language Models for Dynamic Visualisation

Vladimirs Ribakovs

vr112@student.le.ac.uk
239062116

Project Supervisor: Heckel, Reiko (Prof.)
Principal Marker: Goes, Fabricio (Dr.)
Word Count: 12644
(excluding table contents, table labels, headings, references and appendices)

27/04/2025

DECLARATION
All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s). Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s). I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Vladimirs Ribakovs
Date: 14/05/25

Contents

1. Introduction

1.1 Research Question

How can Large Language Models (LLMs) be used to effectively and dynamically process and visualise HESA open-source data to enable the University of Leicester to more efficiently understand its performance against peer institutions?

1.2 Background and context

Introduction to HESA

- Higher Education Statistics Agency (HESA) is the primary and official open-source data collection agency for UK higher education.
- It provides a massive database consisting of millions of records covering various aspects such as institutional performance, funding and benchmarking.
- HESA's mandate is to collect, analyse, and disseminate reliable statistical information about UK higher education.
- The types of data provided by HESA and used in this project include: [5]
    - Student enrolment data (by level, mode, demographic)
    - Staff information (qualifications, contracts, demographics)
    - Financial data (income, expenditure)
    - Facilities information (accommodation, resources)
    - Course and program statistics.

These datasets enable institutions to benchmark against peer institutions and to identify areas for improvement, and track progress over time. [5]

HESA data serves multiple stakeholders:

- Universities use it for strategic planning, performance reviews, and competitive analysis [5]
- Government bodies use it for policy development and funding allocation [5]
- Researchers analyse trends in higher education access, outcomes, and equity [5]

Dynamic Visualisations

Dynamic visualisations are interactive and responsive visual tools used in this project to visualise HESA data. They go beyond static charts by allowing user interaction, real time data exploration, and customization [21]. This project implements it using Chart.js and it allows users to:

- Change the type of visualization on demand.
- Filter data by time periods or institutions.
- Zoom and explore details within charts.
- Compare multiple metrics simultaneously.

This approach transforms raw HESA data into results that can be explored rather than just viewed. Dynamic visualisations are data that is interactively represented based on the user’s submitted

query. It can reveal differently even on the same dataset because it depends on the user's intent. They improve insights by:

- Enabling exploration of underlying patterns.
- Facilitating comparison between different data segments.
- Allowing users to focus on specific areas of interest.
- Supporting different levels of data granularity.

Why using LLMs for this Project

Manual analysis and processing a large volume of complex data is time consuming and difficult, so LLMs offer a solution to this tedious process. In this project, an LLM is used at different stages of the system, such as data processing, analysis, extraction, preparation, recommendations, and even visualization. Even though HESA provides ways to visualise data, but those visualisations are limited and static. Using an LLM can help to work with this data in a more efficient and intuitive way. The Gemini 1.5 Pro was chosen as the LLM for this project because its excellent understanding of natural language queries (NLQ) and translating them into structured operations. This feature addresses challenges in data analytics and allows users without any technical knowledge in this area to access and work with complex datasets.

LLMs offer several advantages for query interpretation:

- Understand context and intent beyond keyword matching
- Handle variations in how questions are phrased
- Recognize domain specific terminology (e.g. "academic years")
- To understand relationships between entities such as institutions, metrics or time periods

Traditional data query methods require knowledge of query languages or tools, understanding of dataset structure, and multiple steps to filter and extract relevant data. LLMs eliminate these barriers by allowing users to simply ask questions in plain English, making data more accessible and reducing the time from question to answer.

The Change of Original Approach

Originally, when the project was just proposed, the GPT-J was selected as the LLM for the project. After exploring GPT-J for some time, I realised that it did not meet the specific requirements of the project, so I had to switch to a more robust and advanced model, Gemini 1.5 Pro. See the section 8.2 on why exactly this approach failed.

Personal Motivation

My personal motivation is to undertake a project that uses an LLM to solve a real world problem. Also, I am fascinated by how much AI (Artificial Intelligence) has improved in the past couple of years, so I want to do more projects using it to get more experience with it. I consider this project a big step toward getting more advanced in working with AI.

1.3 Problem Statement

Current Manual Approach of Processing HESA Data

The current approach to understand the University of Leicester's performance relative to peer institutions using HESA data is manual, tedious, error-prone, and costly [5]. This process requires dedicated staff to spend significant time manually researching and manipulating HESA data.

Even though there is no exact open-source information on how the University of Leicester conducts these searches, based on related information and logical thinking, it can be how it proceeds.

As mentioned above, there is no specific information on how this process is conducted. The University of Leicester does not even say anything about this data being used from the HESA source, at least it is not publicly available.

The University's privacy notice indicates they conduct "statistical analysis" using student data, which suggests that HESA submissions are being used. This strongly suggests that university employees, likely within departments like planning, academic affairs, or institutional research, are involved in this process. [13]

Looking at the job applications that the University of Leicester posts for roles that are closely related to "statistical analyses," I found that they mention the following tools: Power BI, Microsoft Office, or Tableau [14]. According to the discovered information, I can assume that they undertake the following steps to perform current analyses of HESA data:

- Downloading multiple CSV files.
- Cleaning and standardizing data.
- Merging datasets from different years.
- Creating visualizations in tools like Excel, Tableau or Power BI.

Visualisations created through tools like Excel, Tableau, or Power BI further compound these challenges because of inflexible formats. If a research question is changed or followed up by another question, analysts would likely have to restart the entire process – extracting data, reformatting columns, recategorizing values, and generating new charts, which results in significant reporting delays.

Manual methods suffer from:

- Inconsistent data cleaning approaches
- Difficulty in tracking data provenance
- Time intensive repetitive tasks
- Errors in data transformation and aggregation
- Challenges in maintaining and creating visualizations

HESA's Static Visualisations

- Static charts provided by HESA are limited and cannot adapt to users' specific questions.

They cannot show relationships between multiple variables, provide different levels of detail, and support exploration beyond the initial visualization.

- Predefined visualizations cannot be easily changed, so users are restricted to the data and format that HESA provides.
- Users cannot even remove specific data points or change visual attributes like colours or titles.
- HESA offers a variety of charts like line, bar, pie or scatter, but the user cannot decide for themselves what chart they want. Users are forced to use only existing predefined charts.
- Having all these points combined, the fact that users cannot decide what chart to have and cannot change existing charts, it all leads to limited analyses. At best, charts provide a dropdown menu that allows sorting by value, like country, as shown in Figure 1

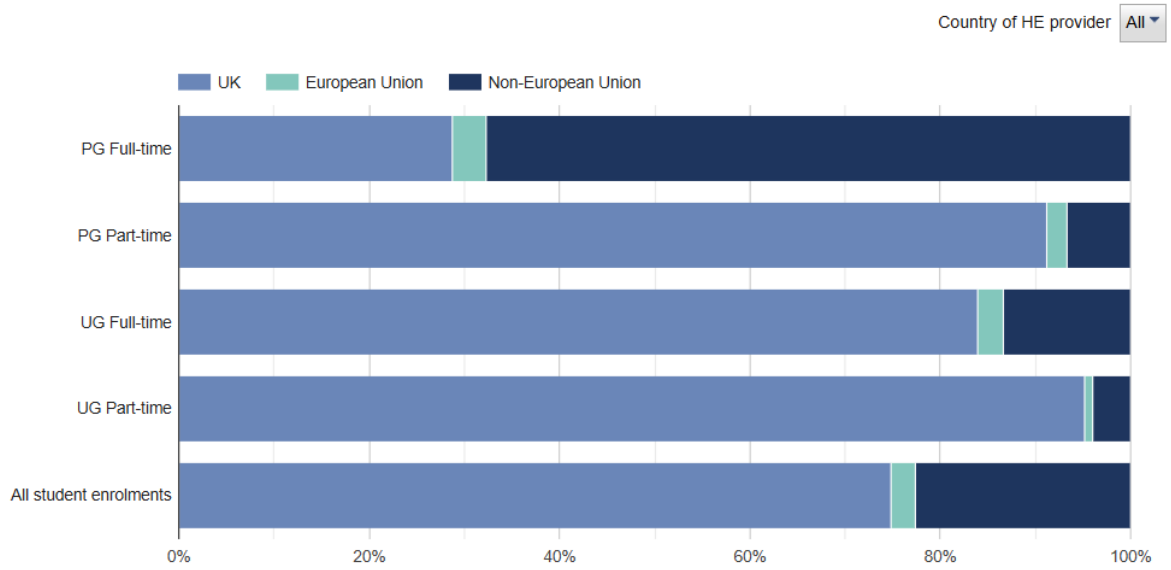


Figure 1: HESA, HE student enrolments by level of study, mode of study and permanent address, Academic year 2023/24.

Inflexibility of HESA Data

HESA does not provide a way to include comparison by Mission Group (Russell Group, University Alliance, Million+), so when an analyst needs to benchmark data against a Mission Group, the data needs to be manually found for each institution in a particular Mission Group. For example, when benchmarking groups shifted to include more institutions from Russell Group, existing reports required manual reconstruction. The inability to dynamically filter, recategorize, or pivot data dimensions without rebuilding entire visualization sets severely limits the exploration of emerging patterns and trends.

1.4 Aims and Objectives

The University of Leicester needs a solution that:

- Adapt to evolving analytical questions

- Provide insights without technical barriers
- Support strategic decision making with relevant comparisons
- Reduce the administrative burden of data analysis.

The project aims to address these limitations by creating a dynamic, user-friendly dashboard powered by an LLM Gemini 1.5 Pro that provides more efficient data comparison, extraction and analysis.

The purpose of this project is to make it easier for the University of Leicester to understand its organisational performance, structure, strengths, and weaknesses against peer institutions by leveraging HESA open-source data. The project aims to transform HESA data interaction through:

- Intelligent data processing that understands academic contexts
- Natural language query interpretation that captures institutional benchmarking needs
- Dynamic visualization that reveals meaningful patterns and comparisons.

The core goal is to reduce the technical and time barriers to analyse HESA data for the University of Leicester. This will allow them to gain powerful insights, ultimately saving time and potentially reducing costs.

1.5 Requirements

Essential:

- **Natural Language Query Interpretation:** The system understands users' free form queries and retrieves and processes data based on that.
- **Integrating and Processing Data:** Extracting and preparing clean HESA data and making it usable for comparing metric performance.
- **Interactive Dashboard:** It must dynamically display results in different formats like tables, charts, diagrams or summary reports.
- **Comparative Analysis**: A feature that must allow grouping and benchmarking universities by different categories like Mission Groups.
- **Data Export Capabilities:** Allow users to download generated reports and data as CSV, PDF or Excel files.

Recommended:

- **Automated Data Retrieval:** Enable automatic updates of the dashboard with new data from HESA when it is available.
- **Using AI To Improve Analysis:** LLM should provide insights, trend identification, and recommendations if user asks for that.
- **Query Building Guide**: Provide a guide or pre-built examples for users who are unfamiliar with the free text queries.
- **Advanced Visualisation:** Support interactive graphs, trend projections that offer visual comparison of performance metrics.

- **Historical Data Tracking:** Tracking data changes over time to see how performance was changing relative to peers.
- **Caching for Frequent Queries:** Implement caching to perform repeated queries more efficient that will result in better performance.
- **LLM Interpretation Feedback:** Allow users to see how their query was interpreted by the system (Pandas) to improved data transparency and remove bias factor.
- **Queries Logs:** Log out user queries to improve future prompts tuning.
- **Data Overload protection**: Implement a safeguard that prevents outputs of big queries to potentially crash the system.
- **CSV File Validation:** Make a mechanism to automatically sanitize new raw HESA data by checking for correct format and size anomalies and then prepare it for the data processing.

Optional:

- **Predictive Modelling:** Using historical data and machine learning (ML) to predict future performance metrics
- **Chatbot Integration:** Include an AI chatbot assistant to help users to refine their queries dynamically, in real time.
- **API for External Use:** Create an API so that external and third-party applications can use dashboard's functionality
- **Collect feedback:** Implement mechanisms to collect users' feedback through the surveys to continuously improve the system
- **Data Storage Scalability**: If HESA data volume grows beyond local CSV storage it will be moved to a more scalable database solution such as PostgreSQL or SQLite.
- **Multiuser Concurrency:** Allow users to submit multiple queries simultaneously by using asynchronous job queues such as Celery and session management.
- **Deployment**: Develop a deployment plan starting with local deployment and extending to containerization such as Docker and orchestration like Kubernetes for future scalability.

1.6 Scope and Limitations

This project focuses on developing a tool for exploring HESA open-source general data in CSV format that is relevant to the University of Leicester.

The project covers:

- Processing and indexing selected HESA CSV datasets.
- Natural language querying of these datasets.
- Automatic visualization generation using Chart.js.
- Comparing and benchmarking institutions that are particularly focused on Mission Groups.
- Provide functionality to export the data.

It does not cover:

- Real time data goes directly from HESA.
- Custom data upload from non-HESA sources.

- Comprehensive processing of all available HESA data.

Why Only University of Leicester Related Data

- The project intentionally focuses on Leicester relevant datasets to provide meaningful benchmarking rather than attempting to cover the entire HESA data.
- Only data that has records about the University of Leicester was downloaded and used for the project because the primary purpose is comparison against the University of Leicester.
- This focused approach provides more relevant and actionable insights for University of Leicester stakeholders. It improves performance by reducing the dataset size, enables better benchmarking and allows for more targeted visualizations.

Only General HESA Data

HESA provides two types of files:

- extremely detailed files containing about 500,000 – 700,000 rows of data on average
- general CSV files that have around 100 – 400 rows of data on average.

The detailed files cause significant challenges for the LLM processing due to token limits restricting how much data can be analysed at once. The restrictions would affect any modern open-source LLM, so this is not a drawback of Gemini 1.5 Pro. Other challenges with detailed data are increased processing time with dataset size, memory constraints for complex analyses, excessive API costs, and often contain redundant or overly granular information. Therefore, this project focuses only on using general HESA data files.

This project uses a total of 392 general HESA CSV files. These general files still provide comprehensive data about various aspects but are less detailed which makes them more suitable for the scope of this project. The project addresses potential challenges of high data volume by implementing selective data processing and indexing functionality. Data is filtered first by academic year(s), then by relevance to the query, and typically only about 10-20 files relevant to a query, where each file has approximately 300 lines, is sent to the LLM for analysis.

This approach is critical for managing processing cost and time. For example, if the system sends 10 – 20 of the detailed files where each file averaging around 600,000 lines of text, this will require processing a significantly larger volume of data. Based on current usage, approximately 227 request cost about £1 of free credit using general files. (see Figure 8)

Note:
Number of requests per £1 = 882 (Total number of request) / 3.88 (Total cost of total number of request) ≈ 227 (see Figure 8)

Using detailed files under similar query conditions would be estimated to cost at least 2000 times more due to the vastly increased data volume processed per query. This highlights the main drawback of using detailed files for dynamic analysis with the current approach and budget constraints. Therefore, this project focuses on using general data to allow for dynamic analysis using the LLM.

Note:
The estimate of "at least 2000 times more" cost when using detailed files is based on the difference in data volume processed per query. The calculation is as follows:

- *Average lines per query with general files: (10 files * 300 lines/file + 20 files * 300 lines/file) / 2 = 4,500 lines*
- *Average lines per query with detailed files: (10 files * 600,000 lines/file + 20 files * 600,000 lines/file) / 2 = 9,000,000 lines*
- *Ratio of detailed to general data volume: 9,000,000 lines / 4,500 lines = 2000 times*

More data per query means much higher token usage, which leads to a significantly higher cost per API call.

1.7 Significance of the Project

This project aims to deliver a significant impact on decision making processes for the University of Leicester. It expects to allow for more flexible and accessible work with the HESA data

The project significantly impacts university decision making by:

- Providing faster access to comparative insights.
- Enabling the University of Leicester (UOL) to explore data without technical barriers.
- Helping the OUL use data to make better decisions
- Supporting real time responses to data-related questions.

These features help with strategic planning by showing patterns and trends that static reports might miss and allowing interactive filtering for deeper analysis.

The project also contributes to automating data visualization and analysis in higher education analytics by:

- Demonstrating effective LLM integration for educational data analysis.
- Creating a reusable framework for natural language data exploration.
- Establishing patterns for automated chart recommendation and generation.
- Providing a feedback loop for continuous improvement.

Long term benefits:

- Reduced administrative burden for data analysis
- More consistent and reproducible data visualization
- Institutional knowledge capture through query patterns
- Improved data literacy across the organization.

1.8 Proposed Solution

The dynamic dashboard is intended to simplify the process of comparing university performance, saving time and potentially reducing costs. It will allow users to query data in plain English and receive dynamic charts, diagrams, and summary reports. Moreover, the dashboard aims to enable benchmarking of the University of Leicester against Mission Groups and other institutions. It would support relatively easy updates as new HESA data becomes available (through the implemented manual upload feature). The motivation is to make comparisons between universities more efficient and accurate. It will allow the University of Leicester to quickly identify its strengths and weaknesses, potentially leading to better resource allocation and improved outcomes.

2. Literature Review

2.1 Overview of HESA Data and Benchmarking in Higher Education

Detailed description of the Structure of HESA Data

HESA data follows a structured format organized by data stream, and it has consistent patterns across different types of information. The system is implemented to understand and process this structure through data processing and indexing functionality. Key data streams relevant to this project include:

- **Students:** Covering entrants, qualifiers, and total numbers [5]. This includes information on:
    - Who is studying in Higher Education (HE),
    - Where they come from, where they study
    - What they study
    - Progression rates and qualifications.
    - Enrolment statistics by level, mode, and year,
    - Demographic information (age, gender, ethnicity),
    - Progression and completion rates,
    - Qualifiers by degree classification
    - Subject area breakdowns.

- **Staff:** Covering demographic and contract characteristics [5]. This includes data on:
    - Who is working in HE
    - What areas they work in
    - What their employment conditions are
    - What their salaries are
    - Where they work
    - Where they come from and go to
    - Employment conditions (contract types, full/part-time)
    - Salary ranges and pay gaps
    - Academic qualifications and specialties

    - Demographic profiles
    - Staff-student ratios

- **Graduates:** Covering survey results showing the activities of recent graduates [5]. This includes data on:
    - Summary statistics on employment outcomes
    - Further study progression
    - Salary information,
    - Industry sectors
    - Graduate satisfaction metrics

- **Finances:** Covering income, expenditure, and financial statements of HE providers [5]. This includes data on:
    - Income and expenditure of HE providers
    - Other financial statements
    - Key Financial Indicators (KFI)
    - Financial data encompasses income sources (tuition, research, commercial)
    - Expenditure categories
    - Balance sheet information
    - Financial sustainability metrics
    - Capital investment patterns

- **Business and Community Interaction:** Covering intellectual property, services, and engagement[5]. This area includes data on:
    - Knowledge exchange activities
    - Patent and IP generation
    - Commercial partnerships
    - Social and community engagement
    - Continuing professional development.

- **Estates Management:** Covering environmental information [5]. This includes:
    - Building condition and space utilization
    - Energy consumption and carbon emissions
    - Waste management and recycling
    - Water usage
    - Sustainability initiatives.

HESA data also appears in multiple formats:

- Tabular data (CSV files with structured rows and columns)

- Time series (sequential data across academic years)
- Categorical breakdowns (data segmented by demographics or characteristics)
- Hierarchical relationships (institutional, departmental, subject levels)
- Geospatial information (regional and institutional locations).

HESA data is the official source for UK higher education information. It provides a standard way to compare institutions and evaluate their performance. This data is essential for university planning, government funding and policy, ensuring quality, following rules, and showing the public how the sector is doing [17].

Reports show that HESA data is key for universities to see how they compare to others. However, there are still problems with getting the data easily, to get it on time, and sharing it to make benchmarking most effective. There are certain rules and difficulties in using numbers to compare institutions [18].

2.2 Large Language Models (LLMs)

Overview of LLMs

LLMs are strong in data projects like this, but they also have downsides [2],[3].
Advantages:

- Understand natural language
- Handle different question styles
- Recognise complex data relationships
- Create an easy-to-read explanation
- Adapt to new topics

Downsides:

- Limits on how much text they can process at once
- Costs for using APIs
- Delays in getting responses
- Sometimes give wrong information
- Need careful setup for prompts.

The Gemini Model (1.5 Pro)

Gemini 1.5 Pro offers significant advantages: very large (undisclosed) parameters, a context window of up to 1 million+ tokens, native multimodality (text, audio, images, video), and accessed via API with associated costs but limited customization compared to self-hosted models [23].

Comparison with Other LLMs (e.g., GPT-J)

Feature	GPT-J	Gemini 1.5 Pro
Parameters	Fewer (Earlier generation)	Significantly More (Undisclosed, but larger)
Context Window	Limited	Vastly Larger (Up to 1M+ tokens)
Multimodality	Primarily Text	Native Multimodality (Text, Audio, Images, Video)
Cost Structure	Free (if self-hosted), Infrastructure cost	API-based (Associated costs, free credits used)
Customisation	More extensive (if self-hosted)	Limited (API-based)
Flexibility	Less flexible	Greater flexibility (Handles diverse data types)
Ease of Use	Requires local setup/hosting	Easy to use (API access)
Performance	Slower processing, higher error rates	Faster processing, better accuracy
Specialised Knowledge	Limited understanding of HE terminology	Superior understanding of HE terminology

Table 1: Comparison of LLM Features (GPT-J vs. Gemini 1.5 Pro) [1], [23]

2.3 LLM Usage in Data Analysis and Visualization

Natural Language to Visualization (NL2Vis)

Natural Language Interfaces for querying and visualising tabular data (NL2Vis) is an active area of research. Recent surveys highlight the significant role of LLMs in this domain [21]. LLMs allow users to interact with data using natural language by converting free form queries into structured formats [22]. This process involves complexities:

- Understanding user intent
- Accurately mapping natural language terms to the underlying data schema
- Clearing up confusion [21].

Handling these challenges allows for automatic data retrieval and chart generation based on user intent [22].

2.4 Data Privacy and Safety with LLMs

Risks and Concerns with LLM APIs

Data safety and privacy are critical issues when working on an application that uses an LLM. This is especially the case when using external APIs. Besides having the general risk of data leakage there are also specific vulnerabilities that exist in the context of LLM applications. However, the HESA data that is being used for this project is publicly available, so it does not have any privacy problems, but this project still implements the safety measures. This could be used in case if the University of Lecester would decide to adapt this project to use sensitive information and create visualizations based on that:

- Prompt Injection: malicious input can manipulate the LLM's behaviour

- Insecure Output Handling: which can lead to exposure of sensitive information processed by the LLM

This is important to understand and follow these specific security measures when building robust and trustworthy LLM powered systems [19].

Mitigation Strategies

Applying threat modelling to the system that uses LLM can help to identify vulnerabilities related to sensitive data handling [20]. The idea for dealing with data privacy risks always applies even if the situation changes. Below principles are focused on protecting user data and maintaining confidentiality [20]. Key mitigation strategies include:

- Keeping Raw Data Local: A fundamental strategy implemented is processing and storing raw HESA data locally. This ensures that sensitive or detailed information is not transmitted to the external LLM API [20].

- Selective Data Processing: Only necessary and anonymized data or metadata is sent to the LLM. Specific tasks like query interpretation or visualization configuration helps with minimizing the exposure of detailed HESA records.

- Robust Input Validation and Sanitization: Implementing careful validation and sanitization of user inputs helps to mitigate risks like Prompt Injection [19].

- Secure API Key Management: Ensuring API keys are stored and used securely to prevent unauthorized access to the LLM service.

- Monitoring and Logging: Implementing logging of LLM interactions can help in identifying and responding to potential security incidents.

2.5 Data Analysis Techniques

Ethical Considerations in Data Analysis

It's very important to consider ethics when collecting, processing, or just analysing data today [15]. HESA data does not publish data that raises ethical questions, but it is important to mention that this project could be used to work with sensitive data if the University of Leicester require it. So, mentioning the importance of ethical concerns would help to avoid the pitfalls with privacy and bias.

Quantitative Data Analysis in Academic Projects

Quantitative data analysis is a basic part of academic projects. There are studies that looked at dissertations and found common problems in how methods and statistics were used [16]. This shows how important it is to use the right research methods and make sure data is processed and

tested accurately. This helps to get reliable and correct results, which makes the analysis of HESA data more trustworthy.

3. Project Design and Planning

3.1 Conceptual Framework and Initial Approach

The project started with a clear focus on improving the way HESA data is being accessed. This is proposed to be done using AI powered natural language processing (NLP). The conceptual structure is based on these three main pillars:

- Data acquisition and preprocessing
- Query interpretation using AI
- Dynamic visualization generation

The original plan was to use AI model GPT-J because of its wide popularity, accessibility and its availability at no cost. Since the model supports self-hosting, it incurs no operational cost to run it on my own hardware. It allows you to maintain control over data processing and enables offline operation for enhanced data security. It supports customization, which is important for higher education specific terms and the model is decent when it comes to query interpretation and dataset matching [1]. Having all these points combined, clearly shows that this model was expected to be great for this type of project. This process followed a sequential flow:

- Data ingestion and preprocessing
- Query analysis using locally hosted GPT-J
- Dataset matching based on extracted parameters
- Visualization generation using predefined templates

Unfortunately, after exploring and using the GPT-J model further, I have concluded that this LLM was not suitable for my project, so I had to switch to a more advanced model like Gemini 1.5 Pro and I explained the reasons in section 8.2.

3.2 Risk Identification and Mitigation Strategies Implemented

I have identified some risks during the project planning phase and developed these strategies to resolve them.

- **Data Integration Complexity:** Having difficulties in merging and cleaning multiple CSV files because of HESA inconsistent formats, missing data, or anomalies in the files. This was mitigated by crafting a robust data cleaning methods using Pandas scripts. The process involves standardizing CSV formats within processing pipeline and performing iterative testing on sample datasets to refine the cleaning logic.

- **Inaccurate Query Interpretation:** To resolve the risk of LLM misinterpreting free text queries, I had to switch to Gemini 1.5 Pro because this model is more capable. I used carefully generated by Geminin and tailored by me prompts that were tailored based on the

HESA data. I created a form of feedback that user could leave about how LLM does its job and implemented caching mechanism to store and quickly reuse successful queries.

- **Performance Bottlenecks with Large Data:** The risk of processing large datasets, which could result in slow query response was mitigated by processing a smaller amount of data by referring to indexing files and using a preview mode that limits displayed data. Also, caching for frequent queries to reduce processing time for repeated requests.

- **Automated Data Update Failures:** The initial risk related to automated data scraping failure was addressed by changing the approach to a manual data upload feature, which is less prone to external website structure changes and provides a reliable method for updating data.

- **User Interface/UX Challenges:** The risk of the dashboard not being intuitive was mitigated by focusing on a clean, minimalist design using Tailwind CSS, implementing interactive features based on usability principles, and planning for user feedback collection to identify areas for improvement.

- **Time Constraints and Scope Management:** The risk of incomplete implementation due to the deadline was managed by prioritizing essential requirements, setting realistic milestones in the revised Gantt charts. Also, important to continuously monitor the progress to adjust the scope if necessary.

- **Data Security and Confidentiality:** The risk of insecure handling of HESA data was mitigated by processing only publicly available open-source HESA data and storing it locally.

Initially, I was working on a backup system that was using regular expressions and math to produce clear query interpretation. This was a part of mitigation strategy in case there would be issue with API or LLM in general. After some time of testing this approach, I realized that I could not get the results I was expecting it to provide me. The main reason it kept failing because of HESA complex data and its diversity so I could not use a pattern to solve it. Even though I tried to cover as many edge cases as possible but even slight variation in the provided data formats or query phrasing could cause the algorithm to fail. Moreover, the eventual success of testing the Gemini API approach made this backup functionality unnecessary.

3.3 Project Phases and Timeline

The project was structured into distinct phases to provide a clear roadmap for development and management. This approach aimed to break down the complex task of building an LLM powered HESA data visualization tool into manageable stages with specific objectives and deliverables.

The initial project timeline was planned to move through the main development areas one by one. The proposed phases were designed to move from basic setup and data preparation through the code system development, AI integration and testing. (see Figure 2)

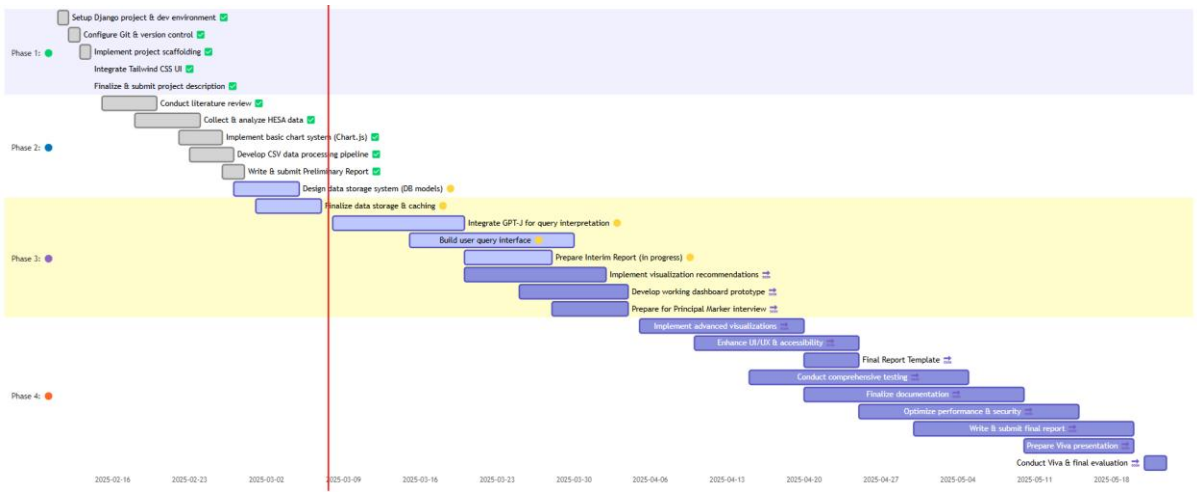


Figure 2: Initial Project Timeline - Gantt Chart

Explanation of Project Phases

- **Phase 1 (Setup & Planning):** Establishing the project foundation, including environment setup, initial research, and defining the project scope and description.
- **Phase 2 (Data & Basic System):** Focusing on acquiring, processing, and storing HESA data, and implementing basic system components like initial visualizations and data processing pipelines.
- **Phase 3 (Core Development & Integration):** Implementing the central features, including LLM integration for query interpretation, building the user interface, and developing the main dashboard functionality.
- **Phase 4 (Finalization & Evaluation):** Refining the system through comprehensive testing, optimizing performance, completing documentation, and preparing for final evaluation.

Phasing breaks the project into manageable stages for better planning, resource allocation, and tracking. It helps reduce risk by allowing for reviews and adjustments at key milestones, ensuring a structured approach to development and completion.

4. System Specification

4.1 Data Flow Diagram

The project is focused on HESA data that is provided as CSV files containing structured data across available sectors. The system creates a comprehensive pipeline that transform raw statistical data into analyses data that is ready to be used. This process includes following: data

collection, metadata extraction, data cleaning and transformation phases. These phases create optimized datasets that powers NLQ system.

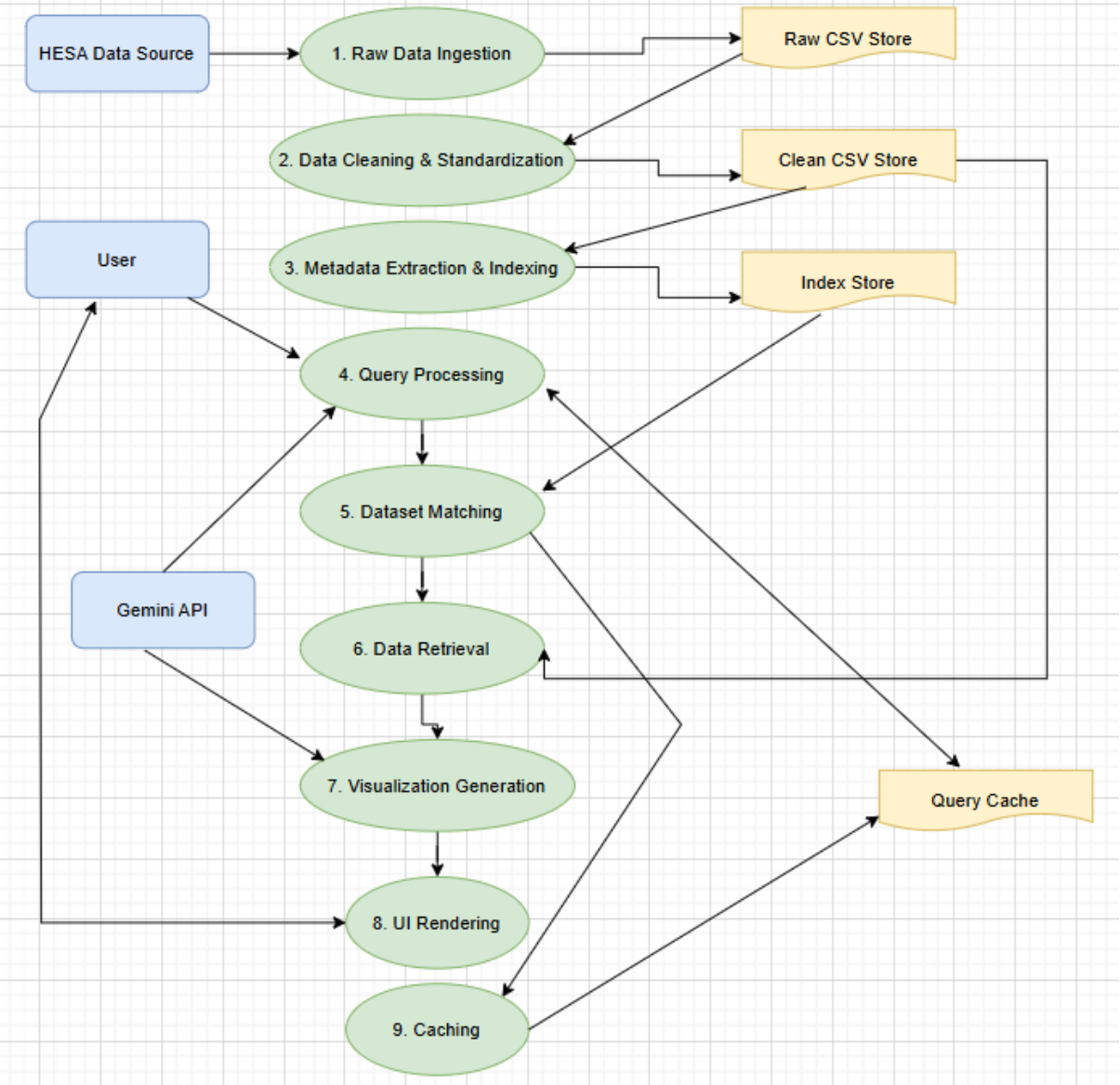


Figure 3: Diagram shows the overall data flow pipeline, from raw HESA data ingestion through cleaning, indexing, query processing, matching, and visualization.

Green ovals = Processes (the steps in the system)
Yellow rectangles = Data stores (where information is kept)
Blue rectangles = External entities (User, HESA, Gemini API)

Process of generating visualization on submitted query:

Source	Destination	Description
User	4. Query Processing	Shows user submitting a query
4. Query Processing	4. Query Processing	Shows processed query parameters moving to dataset matching
5. Dataset Matching	6. Data Retrieval	Shows matched dataset identifiers moving to data retrieval
6. Data Retrieval	7. Visualization Generation	Shows retrieved data moving to visualization generation
7. Visualization Generation	8. UI Rendering	Shows visualization configuration moving to UI rendering
8. UI Rendering	User	Shows results being presented back to the user

Table 2: Explanation of processes to generate visualizations on submitted query

Process of cleaning CSV files:

Source	Destination	Description
Raw Data Ingestion	Raw CSV Store	Shows where processed raw data is stored
Raw CSV Store	2. Data Cleaning & Standardization	Shows raw data being retrieved for cleaning
2. Data Cleaning & Standardization	Clean CSV Store	Shows where cleaned data is stored
Clean CSV Store	3. Metadata Extraction & Indexing	Shows cleaned data being used for indexing
3. Metadata Extraction & Indexing	Index Store	Shows where metadata indexes are stored
Index Store	5. Dataset Matching	Shows indexes being used to match datasets
Clean CSV Store	6. Data Retrieval	Shows where matched datasets are retrieved from
9. Caching	Query Cache	Shows where cached results are stored
Query Cache	4. Query Processing (bidirectional)	Shows checking cache before processing and storing results after

Table 3: Explanation of processes to clean CSV files

Other Processes:

Source	Destination	Description
External Entity: Gemini API	4. Query Processing	This shows the system's interaction with the external AI service

External Entity: Gemini API	7. Visualization Generation	This shows the system's interaction with the external AI service
User	8. UI Rendering (bidirectional)	Shows user providing feedback and modifying visualization parameters

Table 4: Other system processes

4.2 Technical Architecture Design

Technical Architecture Overview

The system employs a modern web application architecture designed to handle data processing, LLM integration, and dynamic visualization.

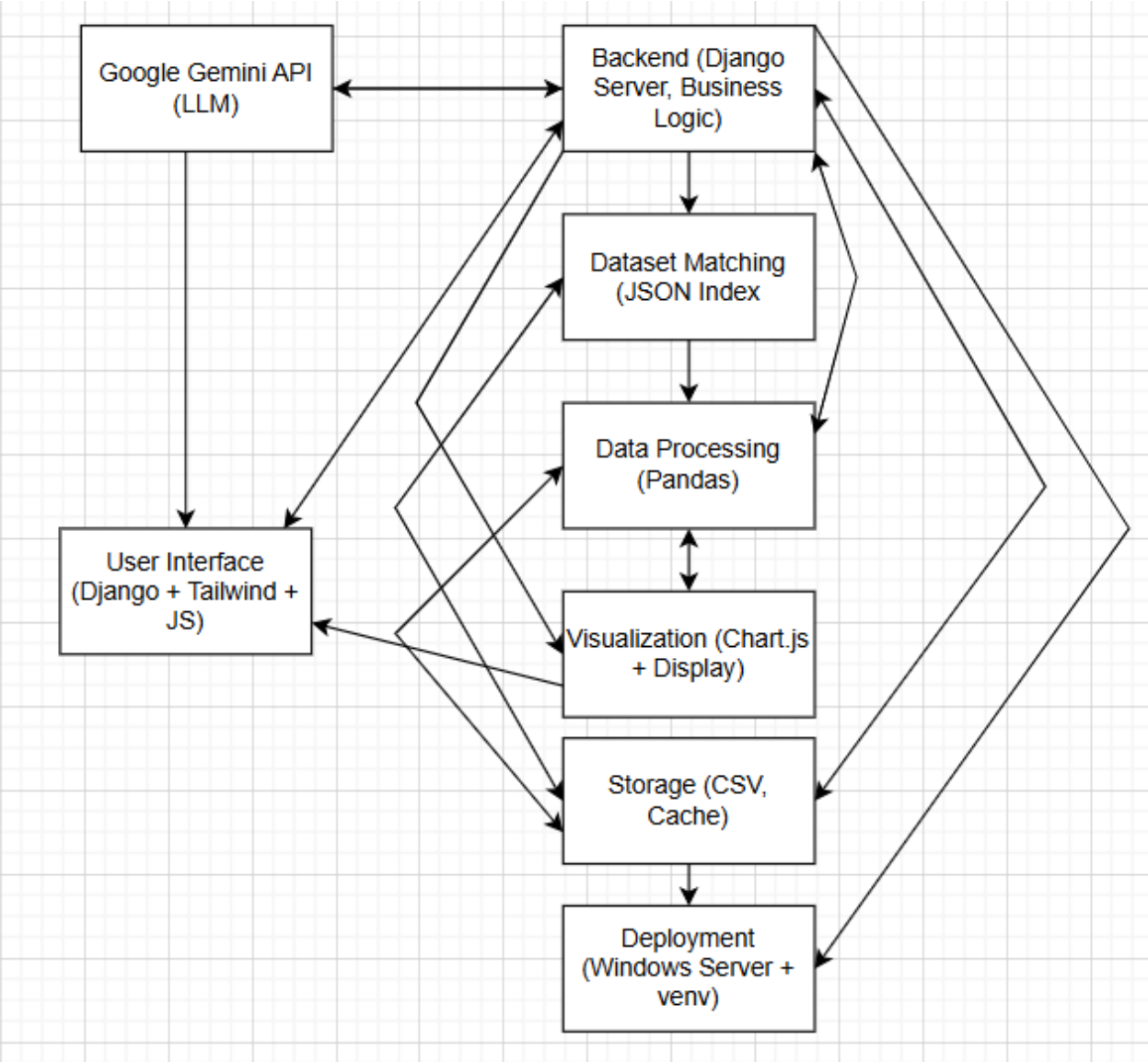


Figure 7: System Technical Architecture Diagram

Source	Interaction	Destination	Description
User Interface	↔	Backend	UI sends natural language queries to backend Backend returns processed results and visualization data to UI
Backend	↔	Google Gemini API	Backend sends user queries to Gemini for entity extraction Gemini returns structured data with institutions, years, and data types
Backend	↔	Storage	Backend reads raw and processed data files Backend writes cache data and user feedback
Dataset Matching	↔	Storage	Reads JSON index file to find relevant datasets Updates index metadata when new files are processed
Data Processing	↔	Storage	Reads CSV data from storage Writes processed results to cache
Data Processing	↔	Visualization	Data processing suggests visualization types based on data characteristics Visualization requests specific data transformations for chart rendering

Backend	↔	Data Processing	Backend sends data processing instructions and query parameters Data Processing returns processed results and analysis summaries
Backend	→	Dataset Matching	Sends extracted entities to identify relevant datasets Forwards query parameters for dataset selection
Backend	→	Visualization	Sends configuration for chart generation Provides display parameters based on query context
Dataset Matching	→	Data Processing	Provides references to matched datasets Sends metadata about dataset structure
Visualization	→	User Interface	Renders charts and data visualizations Provides interactive elements for data exploration
Backend	→	Deployment	Application configuration and settings
Storage	→	Deployment	File system configuration Export to Sheets

Table 5: System Technical Architecture

Development Stack and Dependencies

The project employs a modern web application architecture built on:

Category	Technology	Version	Description
Backend	Django	4.2.5	Web framework

	Python	3.10.12	Primary language
	Pandas	2.0.3	Data manipulation
	NumPy	1.24.3	Numerical processing
Frontend	Tailwind CSS	3.3.3	Styling framework
	JavaScript	ES2021	Client-side functionality
	Chart.js	4.3.0	Visualization library
	HTML5/CSS3	-	Markup and styling
AI Integration	Google Gemini 1.5 Pro API	-	Query interpretation and analysis
	Python google.generativeai	0.3.1	Library
Data Storage	CSV files	-	Primary data storage
	JSON metadata	-	Indexes and cache

Table 6: Project Development Stack and Dependencies

Component Roles and Interactions

The architecture follows a clear separation of concerns:

- **Data Layer:** CSV files store raw and cleaned HESA datasets with JSON metadata for efficient discovery and indexing. Pandas mostly used at the backend to manipulate data (read, clean, transform, and filter)

- **Application Layer:** Django handles requests processing, routing and view rendering. The core functionality is separated into different modules in the Django environment. It includes query processing, dataset matching and configuration of visualization generation.

- **Presentation Layer:** Tailwind CSS provides responsive design through classes. Chart.js renders interactive visualizations based on data processed. The processing happens in the backend and configuration provided via the LLM.

The system follows a request-response cycle where:

1. User queries are submitted from the frontend to Django views.
2. Django views use the data processing pipeline (using Pandas) to access and filter relevant CSV data based on parameters received from the query.
3. The system interacts with the Gemini API to interpret queries, match datasets semantically, and to generate Chart.js visualization configurations.
4. Processed data and visualization configurations are sent back to the frontend.
5. The frontend uses Chart.js to visualize the results in the user's browser.

While architecture focuses on simplicity and maintainability it manages to provide sophisticated data analysis and visualization generations. Pandas is the main dependency that being used throughout the project. It is used for CSV file processing, data cleaning, filtering, transformation and preparation for visualization as shown in Appendices A.1, A.2, A.3.

5. Implementation

5.1 Data Processing Pipeline Implementation

Raw Data Acquisition and Storage

As I explained above in section 1.6, the system is designed to process smaller parts of HESA CSV files to address limitations LLM processing, its performance, memory usage and the cost. This is implemented as a foundational step in the data processing pipeline.

The data ingestion process handles the initial ingestion with validation and basic structure check before being stored. Every raw csv file goes through a check for minimum viability requirements. The file is expected to have a header, columns and basic formatting. (see Appendix D.1)

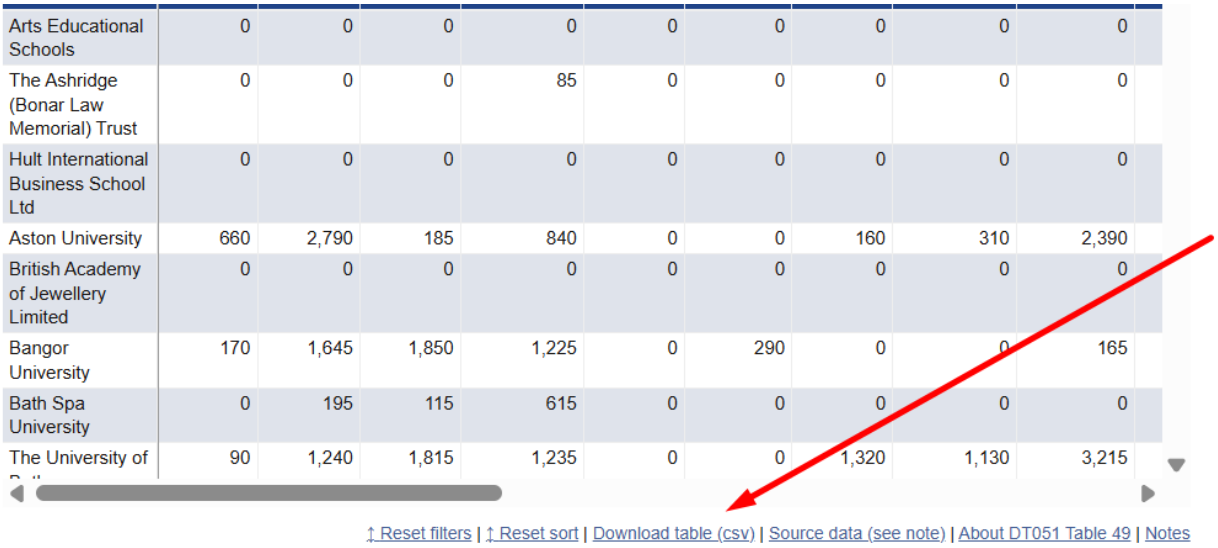
The system support processing of multiple files at the same time. The current dataset provides up to date HESA data as of academic year 2023/24. When HESA provides new data (There is currently as of 2nd of May no exact date of when new data will be released. HESA says "This page lists planned releases for the next six months. We will confirm the exact date of release at least 4 weeks before publication." [24]) user can download it by clicking on the download link on the HESA website of a particular dataset that users want as shown in the Figure 5. When data is downloaded that would look like in Figure 4 the user can upload it to the system. It can be done through the UI (User Interface) at /upload-data/ endpoint and user can get there by clicking on "Upload New Data" in the header section. From there user can select downloaded file/s and click "Upload". (see Appendix D.2)

This manual update system was implemented because HESA does not provide direct API (Application Programme Interface) for automated data retrieval.

```
Title:,"Table 1 - HE staff by HE provider and activity standard occupational classification"
Subtitle:,"Academic years 2014/15 to 2023/24"
Reference ID:,DT025 Table 1
Data source:,HESA
Data source link:,https://www.hesa.ac.uk/data-and-analysis/staff/table-1
Data file canonical link:,https://www.hesa.ac.uk/data-and-analysis/staff/table-1.zip
Licence:,Creative Commons Attribution 4.0 International Licence
Date:,2025
Filters:,Country of HE provider,All
,Region of HE provider,All
,Mode of employment,All
,Academic year,2015/16
Search:,


"UKPRN","HE Provider","Managers, directors and senior officials","Professional occupations","Associate professional occupations","Clerical and manual
"10007783","The University of Aberdeen","0","1,555","10","0","1,565","55","335","425","585","70","75","25","20","255","1,845","3,410"
"10007849","Abertay University","0","205","5","0","210","15","55","65","110","10","5","0","0","65","330","540"
"10007856","Aberystwyth University","0","840","5","0","850","65","305","255","305","85","75","20","15","175","1,305","2,150"
"10000291","Anglia Ruskin University","5","785","15","0","805","60","225","230","425","25","20","15","5","70","1,080","1,885"
"10007759","Aston University","0","675","0","0","675","85","135","240","200","25","25","15","10","120","860","1,535"
"10007857","Bangor University","5","965","20","0","990","60","105","270","400","45","60","0","5","270","1,220","2,210"
"10000571","Bath Spa University","5","605","0","0","610","25","105","105","155","15","25","0","0","75","505","1,115"
"10007850","The University of Bath","0","1,345","0","0","1,345","155","375","345","480","75","65","25","20","370","1,920","3,265"
"10007152","University of Bedfordshire","0","615","20","0","635","15","115","180","260","15","35","15","0","45","680","1,315"
"10005343","Queen's University Belfast","0","1,710","5","0","1,715","30","540","395","735","45","75","15","0","185","2,025","3,740"
"10007760","Birkbeck College","5","1,265","0","0","1,270","50","110","110","280","10","5","15","0","30","610","1,880"
"10007140","Birmingham City University","5","1,675","15","0","1,700","90","245","180","495","15","20","15","5","25","1,095","2,790"
"10006840","The University of Birmingham","5","3,595","30","0","3,635","280","525","1,030","1,105","175","145","35","15","485","3,795","7,430"
```

Figure 4: Raw data file downloaded from HESA . Example file name dt025-table-1.csv.



Arts Educational Schools	0	0	0	0	0	0	0	0	0
The Ashridge (Bonar Law Memorial) Trust	0	0	0	85	0	0	0	0	0
Hult International Business School Ltd	0	0	0	0	0	0	0	0	0
Aston University	660	2,790	185	840	0	0	160	310	2,390
British Academy of Jewellery Limited	0	0	0	0	0	0	0	0	0
Bangor University	170	1,645	1,850	1,225	0	290	0	0	165
Bath Spa University	0	195	115	615	0	0	0	0	0
The University of	90	1,240	1,815	1,235	0	0	1,320	1,130	3,215

Figure 5: Demonstration of how to download dataset from HESA

Metadata Extraction and Indexing

Metadata extraction and the specific extraction logic handled by functions as shown in Appendices D.3, D.4. Those functions analyse files without modifying the underlying data. Functions attempt to extract metadata in multiple ways:

1. Using the processor's method
2. Directly reading the file's first line for a '#METADATA:' prefix
3. Creating minimal metadata from the file content when necessary.

This important step creates a lookup index containing:

- Dataset title and description
- Available academic years (formatted as YY/YY)
- Institutions covered
- Column names and data types
- Value ranges for numeric fields
- Categorical field distributions

The system creates the indexing file *hesa_files_index.json* in the root after processing raw files and collect metadata from all available datasets. (see Appendix D.4). This indexing file helps Gemini to query available datasets, and it consists of:

- File locations (both raw and clean versions)
- Update timestamps for freshness tracking (captured in the 'updated_at' field)
- Metadata about each file
- File references.

This approach allows the system to quickly identify candidate datasets without loading all files for each query, this significantly improves the response time. The metadata extraction processes compile this information into structured JSON file. It supports both exact and semantic matching during query interpretation process. (see Appendix E.2).

This indexing is primarily used to minimize the amount of data being sent to Gemini by sending only data that matches requested academic year and is relevant. This is important because it leads to quicker results and fewer API tokens being used, which potentially reduces cost.

Data Cleaning and Standardization

The CSV cleaning process transforms raw files into datasets that are ready for analysis. This is done through the function that handles the full process from reading the raw files to creating the cleaned output files. As mentioned in the introduction chapter, HESA already provides cleaned and high-quality data, but cleaning process is still required because raw files may contain inconsistencies, different formats and metadata that is not suitable for direct analyses. (see Appendix D.5)

Data sanitization follows a structured approach that includes:

- Header Standardization (extracting and normalizing column names)
- Missing Value Handling
- Academic Year Formatting (using regex patterns)

```
r'(\d{4})[\.\-&_](\d{2})'
```

- Institution Normalization (ensuring consistent naming of institutions)
- Data Type Conversion (using pandas in the extraction process)
- Character Encoding Normalization (handled by opening files with parameters)

```
encoding='utf-8', errors='ignore'
```

- Comprehensive metadata extraction

The file naming convention for cleaned files is described in Appendices D.5, D.6. The logic includes:

- Title from the raw file (extracted in the code using regex patterns)

```
re.search(r'Table\s+\d+\s*-\s*(.*)', title_text)
```

- Academic year(s) covered (extracted in the code using regex patterns)

```
re.search(r'(?:,|\:)\s*Academic year\s*,\s*(20\d{2}/\d{2})', line)
```

- Original raw file name

For example, a raw file as shown in Figure 4.

- file name: dt025-table-1.csv
- title: HE staff by HE provider and activity standard occupational classification
- academic year: 2015/16

After cleaning, the file will be named: HE staff by HE provider and activity standard occupational classification 2015-16 dt025-table-1 (1).csv (see Figure 6)

This file name convention is necessary because HESA sometimes provide different datasets but with same academic year and even title. It is vital to make the original name when creating a unique identifier for the cleaned file. The fact that CSV files can have same titles along with academic years represent a challenge since it requires the cleaning process to be more detailed to prevent possible duplications.

```
#METADATA:{"title": "HE staff by HE provider and activity standard occupational classification", "academic_year": "2015/16", "keywords_title": ["staff", "provider", "activity", "standard", "occupational", "classification"], "keywords_columns": ["provider", "managers", "directors", "senior", "officials", "professional", "occupations", "associate", "clerical", "manual", "total", "academic", "staff", "administrative", "secretarial", "skilled", "trades", "caring", "leisure", "other", "service", "sales", "customer", "process", "plant", "machine", "operatives", "elementary", "non-academic"], "original_filename": "dt025-table-1 (1).csv", "new_filename": "HE staff by HE provider and activity standard occupational classification 2015-16 dt025-table-1 (1).csv"}
HE Provider,"Managers, directors and senior officials",Professional occupations,Associate professional occupations,Clerical and manual occupations,Total academic staff,"Managers, directors and senior officials",Professional occupations,Associate professional occupations,Administrative and secretarial occupations,Skilled trades occupations,"Caring, leisure and other service occupations",Sales and customer service occupations,"Process, plant and machine operatives",Elementary occupations,Total non-academic staff,Total
The University of Aberdeen,0,"1,555",10,0,"1,565",55,335,425,585,70,75,25,20,255,"1,845","3,410"
Abertay University,0,205,5,0,210,15,55,65,110,10,5,0,0,65,330,540
Aberystwyth University,0,840,5,0,850,65,305,255,305,85,75,20,15,175,"1,305","2,150"
Anglia Ruskin University,5,785,15,0,805,60,225,230,425,25,20,15,5,70,"1,080","1,885"
Aston University,0,675,0,0,675,85,135,240,200,25,25,15,10,120,860,"1,535"
Bangor University,5,965,20,0,990,60,105,270,400,45,60,0,5,270,"1,220","2,210"
Bath Spa University,5,605,0,0,610,25,105,105,155,15,25,0,0,75,505,"1,115"
The University of Bath,0,"1,345",0,0,"1,345",155,375,345,480,75,65,25,20,370,"1,920","3,265"
University of Bedfordshire,0,615,20,0,635,15,115,180,260,15,35,15,0,45,680,"1,315"
Queen's University Belfast,0,"1,710",5,0,"1,715",30,540,395,735,45,75,15,0,185,"2,025","3,740"
Birkbeck College,5,"1,265",0,0,"1,270",50,110,110,280,10,5,15,0,30,610,"1,880"
Birmingham City University,5,"1,675",15,0,"1,700",90,245,180,495,15,20,15,5,25,"1,095","2,790"
The University of Birmingham,5,"3,595",30,0,"3,635",280,525,"1,030","1,105",175,145,35,15,485,"3,795","7,430"
University College Birmingham,0,290,0,0,290,20,50,35,80,15,10,15,0,80,300,590
Birmingham Newman University,0,155,5,0,160,15,20,25,65,5,5,0,0,35,175,335
Bishop Grosseteste University,0,95,0,0,95,15,30,40,60,5,30,0,0,15,190,285
The University of Bolton,0,300,25,0,330,15,60,70,80,5,15,0,5,50,305,635
The Arts University Bournemouth,0,350,0,0,355,15,30,55,40,0,5,0,0,0,145,500
Bournemouth University,0,935,0,0,935,50,250,300,255,10,35,0,0,10,905,"1,845"
```

Figure 6: Example of a cleaned csv file name based on the raw csv file Figure 4.

Transformation and Enhanced Metadata Generation

Cleaned data files are changed into optimized formats for analysis using a data processing system. We also get extra information (metadata) to help with this. (see Appendix D.3, D.4)
This process prepares the data by:

- Calculating metrics for analysis
- Structuring it for clear visualization
- Adding helpful context

Enhanced metadata is generated for cleaned files through the metadata extraction process. It extracts information like titles, academic years and columns while handling potential inconsistencies and missing data. The system tries multiple approaches to extract metadata and if it does not work, it falls back to more basic method (the basic method serves as fallback mechanism but so far, I have not faced this issue). (see Appendix D.4)

The process adds analytical context beyond basic structure:

- It identifies academic years using multiple pattern-matching techniques
- Extracts column information that can be used for visualization type recommendations
- Creates relationships between datasets by incorporating all this information into a central index file that can be queried for cross-analysis.

Rationale for Separate Raw and Clean Files

This dual storage approach provides several advantages: (see Appendices D.5, D.6, D.4)

- Data integrity
  - Original HESA files remain untouched in the raw_files directory)
- Processing efficiency
  - Clean files in the clean_files directory optimize structure for analysis
- Versioning support
  - Changes to cleaning methodologies can generate new clean versions through reprocessing
- Failure recovery
  - Processing errors can be addressed by regenerating clean files from the source
- Selective enhancement
  - metadata can be progressively enhanced for clean files through the multiple extraction methods implemented

It keeps track of the relationship between raw files, processed output and metadata (inside the *hesa_files_index.json*). This ensures that the data is changed using the same standard rules for all HESA information. This full process helps us to protect the original data and prepare it well for analysis and charts generation.

5.2 Query Processing and Interpretation

Overview of Natural Language Query Interpretation

The system processes natural language queries (NLQ) using a sophisticated pipeline. It leverages Gemini 1.5 Pro to transform free-text queries into structured parameters. It sends carefully created prompts to Gemini API along with system instructions that explains to the AI how to extract structured data from NLQ. (see Appendices L, E.1)

This AI powered approach recognizes query formulations like “Show me student enrolment at Leicester in 2019” or “What was the staff-student ratio for Russell Group universities between 2015-2018?”. The method identifies query intent by constructing a detailed system prompt which instructs AI to extract institutions, academic years and user requested data. It preserves semantic relationships through the structured JSON response format that captures relationship between extracted entities.

The functionality understands different types of queries through its complex prompt engineering. As mentioned above, this includes:

- Instructions for handling academic year conventions
- Institution name variations
- Data request specifications.

It adapts the response format according to the query type and returns structured data that can be used by other application functionalities. The implementation shown in Appendix E.1 along with

integration with the Gemini API enables intuitive data exploration without the need for users to learn specific query syntax

Parameter Extraction, Tokenisation, and Normalization Techniques

The parameter extraction process as shown in Appendix L involves:

- Tokenization of queries into meaningful segments
- Entity recognition for key parameters (institutions, years, metrics)
- Normalization of recognized entities to standard formats
- Validation against expected patterns and ranges, and structuring for dataset matching.

Parameter extraction does not directly tokenize the query itself but rather constructs a detailed prompt that instructs the AI to perform entity extraction. Using this method, the system gets detailed instructions (the prompt) on how to find and sort the key information, such as institutions, time periods and comparison groups. The method includes a step to process the AI's answer. This step takes the information and transform it into the exact formats the system needs. (see Appendix E.1).

One of the most important aspects of this process is to accurately interpret the academic year which spans two calendar years. The system uses a custom function detailed in Appendix E.2 that applies higher education-specific logic to correctly interpret various date references:

- "Starting in 2017" or "from 2017" → Matches the 2017/18 academic year
- "End of 2017" or "ending in 2017" → Matches the 2016/17 academic year
- Plain years (e.g., "in 2017") → Treated as the starting year (2017/18)
- Year ranges (e.g., "2016 to 2017") → Matches the 2016/17-2017/18 academic years
- "Past 5 years" → Matches the last 5 academic years from the current year

This logic ensures that the data-based queries are correctly interpreted and ready for further use.

LLM Integration Design

The Gemini 1.5 API integration is shown in Appendices E.1, K, L.
It follows a structured design approach:

- Prompt Engineering
    - Designing effective prompts for Gemini that include the user query, relevant context (like the structure of available data), and clear instructions for entity extraction, intent classification, and relevance scoring.

- API Client Implementation

  A robust API client was developed to communicate with the Gemini API. It includes initialization with API key, tailored request formatting, and extensive error handling with detailed logging. It uses the requests library for making HTTP

requests.

- Response Parsing and Validation
    - Designing logic to parse the structured JSON response from Gemini and validate its content to ensure it can be used by the subsequent processing steps.

- Caching Integration
    - Designing the query processing pipeline to check the cache before sending a request to the Gemini API, and to store the results upon receiving a valid response.

The Gemini 1.5 Pro API was configured with specific parameters to ensure optimal performance and output for the project's needs as shown in Appendix L. Key settings include:

- Output Capacity
    - Configured to generate up to 2,048 tokens per response based on the configured parameter *maxOutputTokens: 2048*. Balances detailed output with managing token usage.

- Temperature Setting
    - Set to a low value of *temperature: 0.1*. This makes responses more deterministic, focused, and less random. This is important for accurate data interpretation and structured output generation.

- Top-P and Top-K
    - Using *topP = 0.95 and topK = 40*. These parameters are responsible for the randomness of the output. This ensures a balance between creative interpretation and following to the provided context. This helps with reliability of entity extraction and data matching.

The API integration flow detailed in Appendix L includes:

- Preparation of well-structured prompts with necessary context (system_prompt and user_prompt)

- Secure API communication with appropriate authentication

```
headers = {"Content-Type": "application/json", "x-goog-api-key": self.api_key}
```

- Response validation and error handling (with multiple error checks and try-except blocks)

- Parsing of structured information from responses (JSON extraction and validation)

- Seamless integration with the existing processing pipeline.

5.3 Visualization Generation

Chart Generation

It analyses the selected dataset and user request, then calls the Gemini using the client to determine the correct chart type. After the type is configured, it generates Chart.js configurations for rendering the determined chart. (see Appendix F.1)

The implementation includes

- Detailed processing of dataset information
- Construction of prompts for the Gemini API
- Extraction of chart configuration from the AI response.

Chart Types Implementation

The system prepares data for different chart types like line, bar, and pie charts. This functionality is implemented across several methods (see Appendices F.2, F.3, F.4) including:

- Handles different chart type requests
- Gets AI recommendations for chart types
- Allows dynamically changing the visualization

Visualization generation uses hybrid approach. Gemini generates Chart.js configuration by defining chart type, its structure, labels and options. The generation is based on the dataset characteristics and user submitted request. The whole configuration is then sent to the front-end where the actual dataset data which is fetched and [processed by the backed is combined with the configuration. Lastly, after it gets combined the Chart.js renders the interactive chart for the user on the webpage. The importance of this approach is that it ensures data privacy because the actual data does not go to Gemini thus, technically this approach can be used with other data that is more sensitive. (see Appendix F.1)

Frontend Rendering

The client-side rendering instantiates Chart.js visualizations on canvas elements. Interactive controls are managed for creating new charts and switching visualization formats. Event listeners on UI elements connect user actions to these functions which enables dynamic updates based on user queries and selections. (see Appendices G.1, G.2, G.3)

Interactive Visualization Controls

The system allows users to toggle data visibility, swap axis and adjust colour directly on the charts. The control is implemented in the frontend JavaScript and connected to the backend functions that handles data transformation request. (see Appendices G, H.1)

Real time Chart Regeneration

Generated charts are updated automatically when filtering parameters or visualization options changes. This created using the API endpoint and AJAX request in the frontend. (see Appendix H.1)

Visualization Type Recommendations

The system uses AI to suggest the best chart type for visualizing data. It analyses the dataset and explains why specific chart type is recommended therefore helping user to understand effective data visualization. (see Appendices F.3, H.2)

5.4 UI Implementation and Interactivity

Dashboard Interface

The interface is implemented with key functions including: (Appendices I.1, I.2, I.3, I.4)

- Handles displaying query results
    - Transforms JSON responses into user-friendly HTML, including formatting institution names, academic years, and rendering clickable dataset cards.

- Creates reviewable dataset cards
    - Generates interactive cards for each matched dataset, displaying title, metadata, and sample rows with clean formatting.

- Manage loading states
    - Provides visual feedback during API calls, displaying animated spinners and contextual messages about current processing steps.

- Event listeners for query input
    - Captures user inputs, validates queries, and submits requests to backend APIs with appropriate feedback mechanisms.

Dataset Details View

Implemented with functions including:

- Displays interactive data visualizations (see Appendix G.1)
    - Renders dynamic charts with hover effects, tooltips, and interactive legends that allow users to isolate specific data series.

- Creates charts based on user specifications (see Appendix G.2)
    - Processes natural language visualization requests and transforms them into appropriate data visualizations, considering dataset characteristics for optimal presentation

- Allows switching between visualization formats (see Appendix G.3)
    - Enables seamless conversion between chart types (bar, line, pie) while preserving the dataset context and user's visualization intent.

- PDF and CSV export functionality implemented through dedicated download handlers (see Appendix I.5)
    - Provides client-side export capabilities that format data with proper headers, styling, and metadata for professional reports and further analysis.

Dynamic Dataset Previews

The system displays the first three rows of matched datasets with collapsible sections for expanded metadata so users can quickly find relevant dataset.

Data previews are implemented with functions to:

- Prepare data for clean display (see Appendix I.6)
    - Removes technical artifacts like metadata headers and normalizes column names for better readability. It helps to ensure consistent presentation across different dataset types.
- Toggle functionality through (see Appendix I.7)
    - Enables users to expand and collapse detailed information sections with smooth transitions. It keeps the screen clean and allows the user to access that extra details when needed.
- Collapsible metadata sections with animation classes

```
animate-fade-in, animate-fade-out
```

    - Creates a polished user experience with subtle transitions when revealing or hiding content, improving interface responsiveness and visual feedback.

Responsive Design

The interface adapts to various screen sizes while maintaining data integrity and usability.

The interface uses Tailwind CSS classes throughout the templates with responsive design patterns including:

- Flexible grid layouts

```
<div class="grid grid-cols-1 md:grid-cols-3 gap-4">
  <div class="border rounded p-3 bg-white">...</div>
  <div class="border rounded p-3 bg-white">...</div>
  <div class="border rounded p-3 bg-white">...</div>
</div>
```

- Responsive padding and margins

```
div class="max-w-7xl mx-auto px-4 py-8 md:px-6 lg:px-8">
  <h1 class="text-2xl md:text-3xl font-bold mb-4 md:mb-6">HESA AI Powered Data Dashboard</h1>
</div>
```

- Overflow handling for data tables

```
.table-container {
  overflow-x: auto;
```

```
  max-height: 300px;
  overflow-y: auto;
  scrollbar-width: thin;
}
```

- Mobile-friendly controls with appropriate sizing and spacing

```
<button
  class="w-full sm:w-auto px-3 py-2 sm:px-4 sm:py-3 text-sm sm:text-base bg-blue-600 text-white rounded-md flex items-center justify-center"
>
  <span class="flex items-center">
    <svg class="h-4 w-4 mr-2" viewBox="0 0 24 24">...</svg>
    <span>Search</span>
  </span>
</button>
```

5.5 Feedback System

Feedback Collection

It provides an interface for user to rate received results with options as “Not Helpful”, “Helpful” and “Very Helpful”. Optionally, displays a comment area for detailed feedback if user selects anything but the option “Very Helpful”. (see Appendix J)

Feedback Submission

The system collects relevant data such as:

- Query details
- Dataset information
- Visualisation settings
- User rating
- Comments

This data is sent to the server for storage and analysis so it can be used further to enable data-driven improvements. (see Appendix J)

5.6 Caching and Performance Optimization

Query Cache Implementation

The system implemented a caching functionality that stores and retrieves query results as JSON documents in */cached* folder. This caching system uses hash-based keys and metadata tracking to avoid redundant API calls and data processing. The cached data includes all the information about the query.

The cached data lasts for 30 days so after it expires the system removes it from the folder. This allows to maintain the system effectively without manual clean up. Also, it tracks usage statistics

like access accounts and last accessed timestamp. These statistics can be used to make performance analyses and optimization of frequently used queries. (see Appendix K)

5.7 Codebase Structure and Endpoints

Codebase Overview

hesa-llm-visualisation/

```
├── core/                          # Main application directory
│   ├── templates/                 # HTML templates
│   │   ├── ai_dashboard.html      # AI powered dashboard interface
│   │   ├── base.html              # Base template with common layout
│   │   ├── documentation.html     # Project documentation page
│   │   ├── upload_data.html       # Data upload interface
│   │   └── visualization/         # Chart visualization templates
│   │
│   ├── static/                    # Static files (CSS, JS, images)
│   │   ├── css/                   # Compiled CSS
│   │   ├── js/                    # JavaScript files
│   │   │   └── ai_dashboard.js    # Main dashboard functionality
│   │   └── src/                   # Source files for CSS
│   │
│   ├── utils/                     # Utility functions
│   │   └── query_cache.py         # Query caching implementation
│   │
│   ├── services/                  # Service layer
│   ├── data_processing/           # Data processing modules
│   ├── visualization/             # Visualization generation code
│   ├── management/                # Django management commands
│   │
│   ├── views.py                   # Main view controllers (5300+ lines)
│   ├── urls.py                    # URL routing
│   ├── gemini_client.py           # Google Gemini API integration
│   ├── data_processor.py          # Data processing logic
│   ├── data_processing.py         # Additional data processing utilities
│   ├── models.py                  # Data models (minimal)
│   └── apps.py                    # App configuration
│
├── hesa_llm_visualisation/        # Project configuration
│   ├── settings.py                # Django settings
│   ├── urls.py                    # Project URL routing
│   └── wsgi.py                    # WSGI configuration
│
├── data/                          # Data storage directory
│   ├── raw_files/                 # Original HESA data files
│   └── cleaned_files/             # Processed data files
```

```
├── docs/                          # Documentation files
│
├── manage.py                      # Django project management script
├── debug_metadata.py              # Debugging utilities
├── csv_cleaning.py                # CSV file cleaning utilities
├── hesa_files_index.json          # Index of available HESA data files
│
├── run_django_server.bat          # Windows batch script to run server
├── data.env                       # Environment variables
│
├── package.json                   # Node.js dependencies
├── tailwind.config.js             # Tailwind CSS configuration
├── postcss.config.js              # PostCSS configuration
└── README.md                      # Project documentation
```

Key Endpoints

Below are key endpoints that user can access and easily explore. There are other endpoints that are not for user interaction but rather for the usage of Gemini to return or display data.

- / or /ai-dashboard/ (*views.ai_dashboard*)
  - Main entry point to the application
- /documentation/ (*views.documentation*)
  - Documentation page
- /process_gemini_query/ (*views.process_gemini_query*)
  - Processes natural language queries using Google Gemini
- /ai_dataset_details/ (*views.ai_dataset_details*)
  - Gets detailed information about datasets
- /save_feedback/ (*views.save_feedback*)
  - Saves user feedback about query results
- /upload-data/ (*views.upload_data_view*)
  - Data upload interface
- /process-upload/ (*views.process_upload*)
  - Processes uploaded data files

5.8 System Setup and Running Instructions

Please refer to the STARTUP.md file inside the submitted project.

Location: vr112\STARTUP.md

It has full instruction on how to set up the project and run it

6. Demonstration

6.1 Using the AI Powered Query Interface

Submitting a Natural Language Query

Main Dashboard (/ or /*ai-dashboard*)

1. Navigate to the main dashboard (see Figure 8)
    - Open the application in the browser at the root URL
    - You will see the "HESA AI Powered Data Dashboard" with a search input field

2. Enter a NLQ in the search box (see Figure 9)
    - Query example: "How many postgraduates studied at the University of Leicester between 2018-2020?"
    - The query input field displays a placeholder example to guide the user

3. Configure search parameters (optional)
    - Set "Number of matches to show" (by default: 3)
    - Select a Mission Group filter if needed. Options: None. Russell Group, Million+ or University Alliance

4. Click the "Search" button (see Figure 10)
    - The system will process query using Gemini 1.5 Pro API
    - A loading indicator will appear during the process

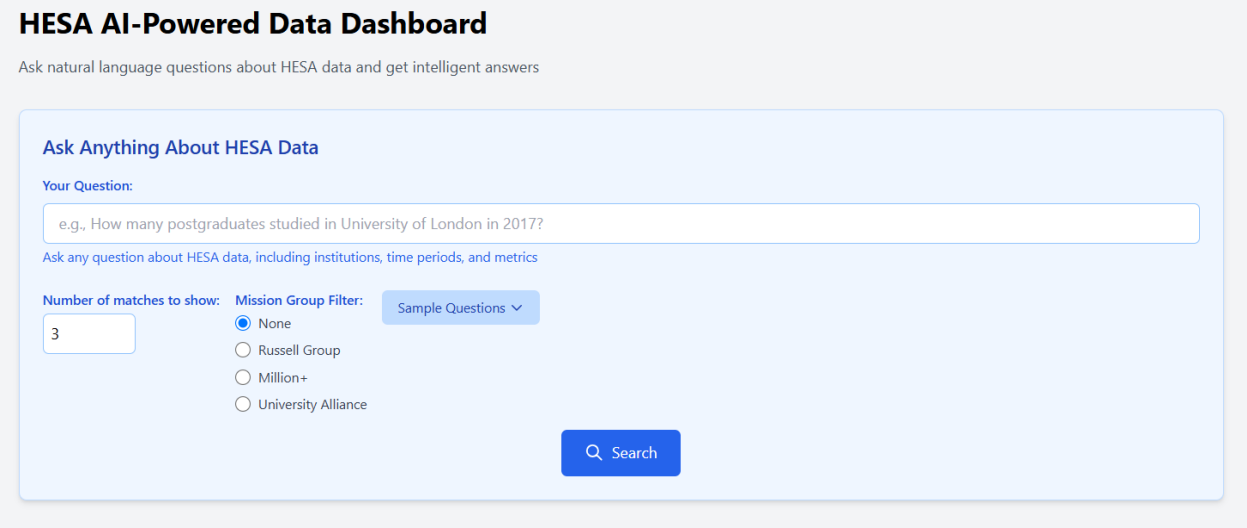


Figure 8: Main Dashboard webpage

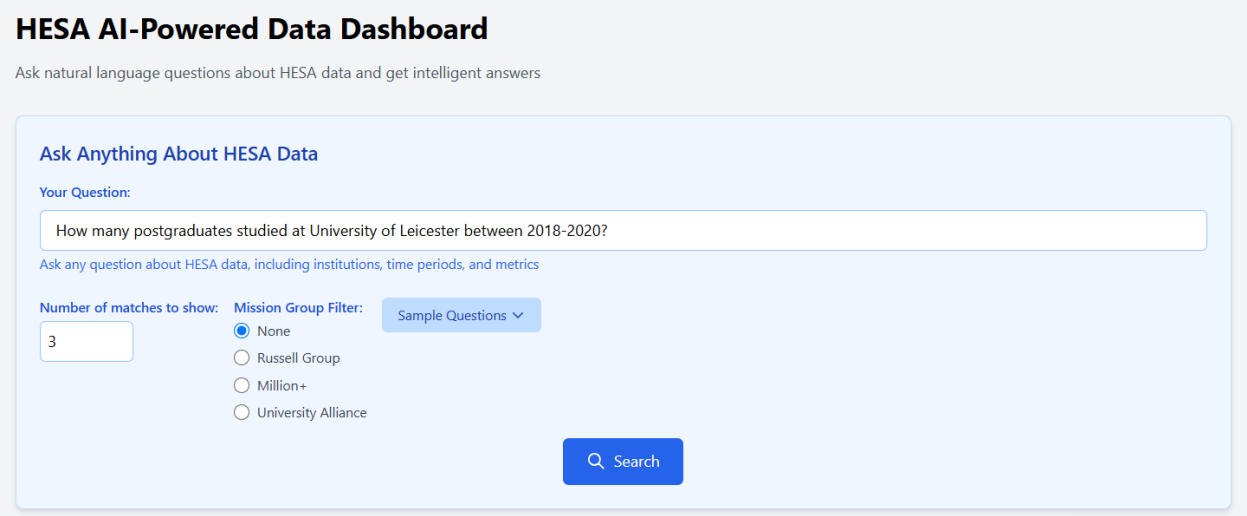


Figure 9: Main Dashboard webpage with query example

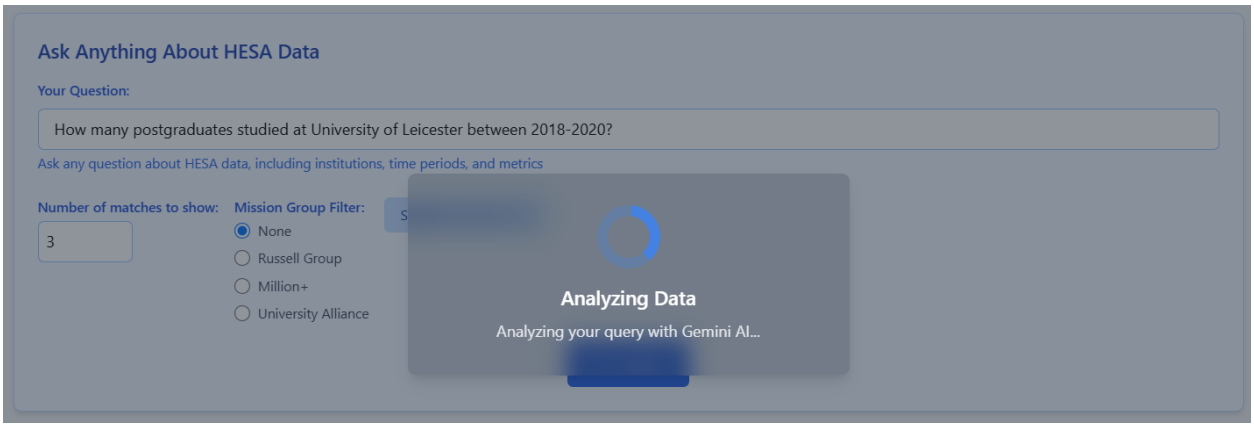


Figure 10: Indicator of submitted query

6.2 Query Interface

Reviewing Query Analysis Results

Endpoint: */process_gemini_query* (called in background)

1. View query interpretation (see Figure 11)
    - The system displays how it understood the query
    - Shows extracted institution (“University of Leicester”)
    - Shows extracted time period (“2018-2020”)
    - Shows extracted requested data (“postgraduate students”)

2. Review matching datasets (see Figure 12)
    - Each matching dataset shows score of relevance (percentage match)
    - Datasets are ranked by relevance score to the submitted query
    - Preview of data columns and rows is provided

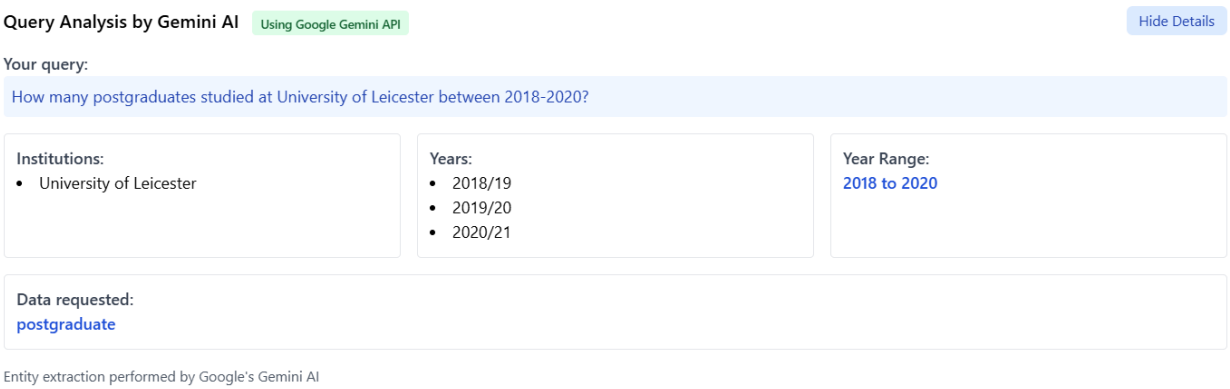


Figure 11: Query Interpretation information

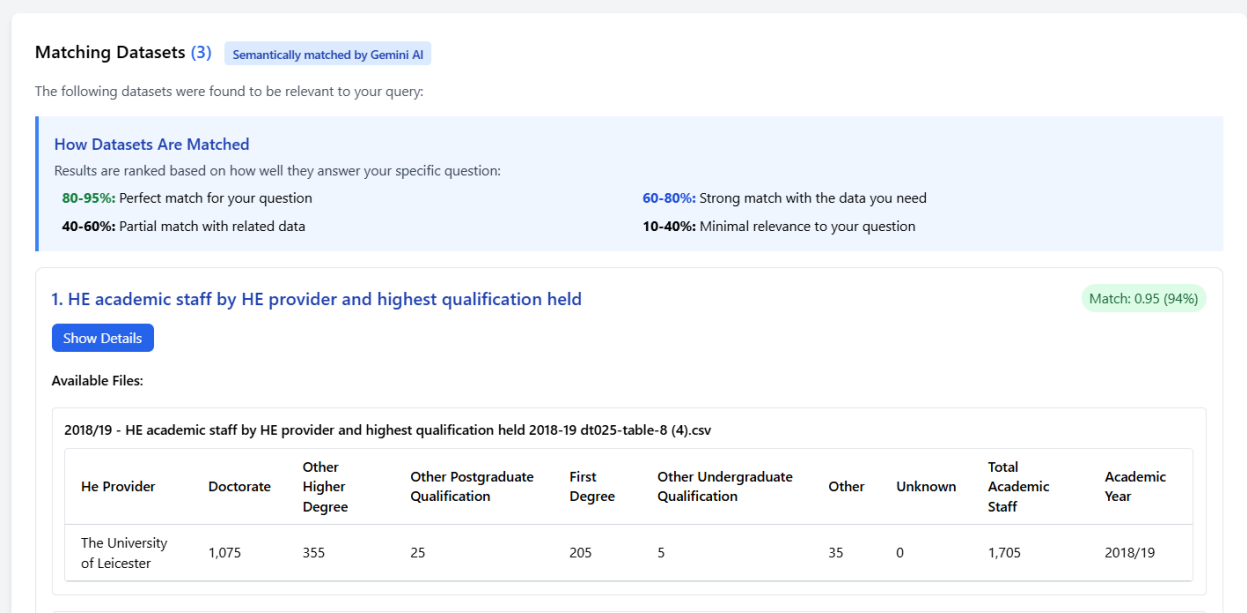


Figure 12: Matched datasets previews and datasets matching information

Working with Dataset Details and Visualization

Endpoint: */ai_dataset_details* (called in background)

1. Click on a dataset from the found results (see Figure 13)
    - Look for most relevant dataset for submitted query
    - Review dataset details including time period covered and available institutions
    - Click “View Details” or “Explore” button for the chosen dataset

2. Review detailed dataset information
    - View metadata about the dataset
    - See all available columns and preview rows
    - Data is filtered based on query parameters (institution and years)

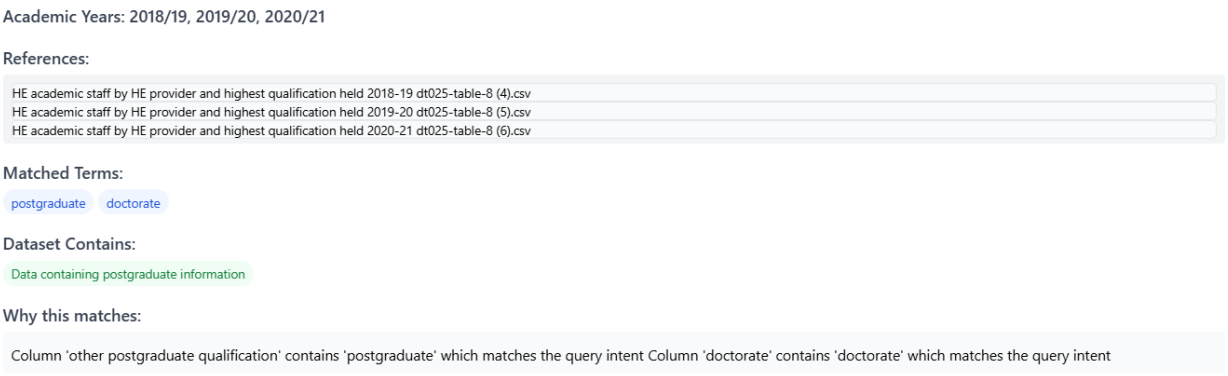


Figure 13: Information about why Gemini selected this dataset

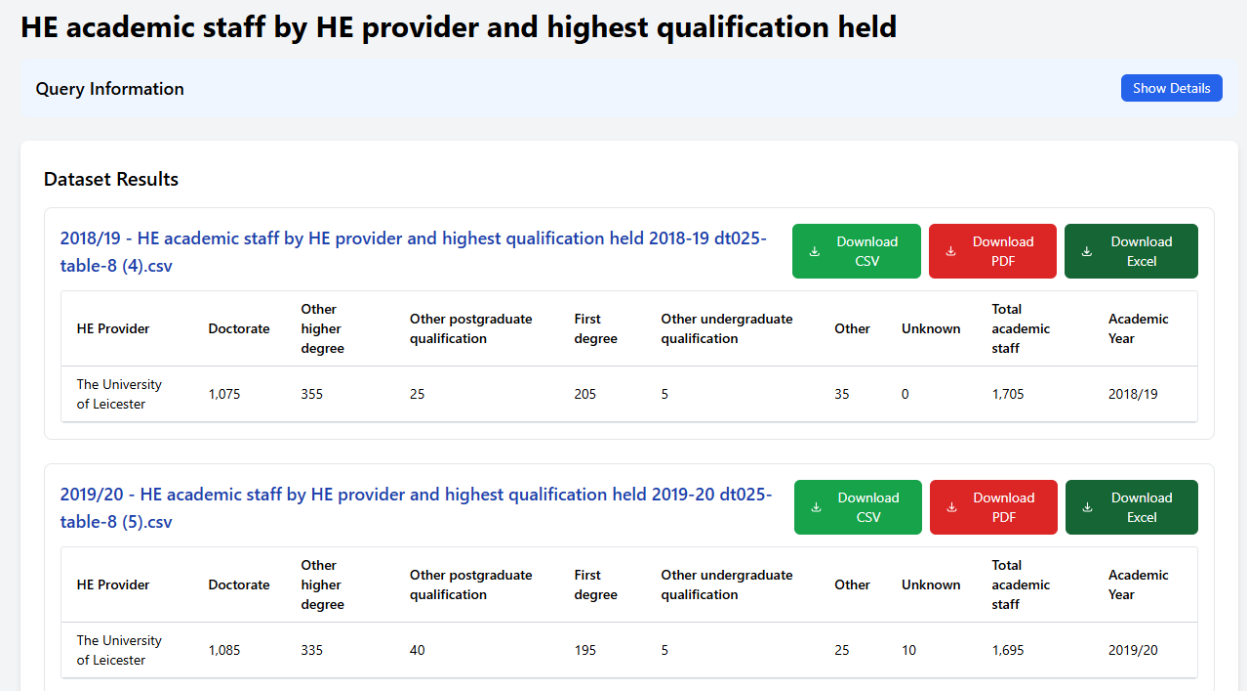


Figure 14: A selected dataset

Generating and Interacting with Visualizations

Endpoint: */visualization_api* (called in background)

1. Analyse visualisation recommendation (see Figure 15)
    - The system uses Gemini to automatically suggest the most appropriate chart type for selected dataset
    - Example: Line chart for time-series data showing trends over multiple years
    - A brief explanation of why selected type of chart is recommended

2. Customize visualization (see Figure 17)
    - Enter a specific visualisation request (“Show a comparison of postgraduate numbers by year”)
    - Select different chart types from options (bar, line and pie)
    - View explanations for each chart type to understand its strengths and limitations

3. Generate visualisation (see Figure 17)
    - Click “Generate Visualization” button
    - The system will create an interactive chart based on chosen specifications
    - Chart will appear with proper titles, label, and colour

4. Interact with generated visualization (see Figure 18,19)
    - Hover over data points to see exact values
    - Toggle data series on/off (for multi-series charts)
    - View the visualization in full screen mode if desired
    - See insights of generated charts

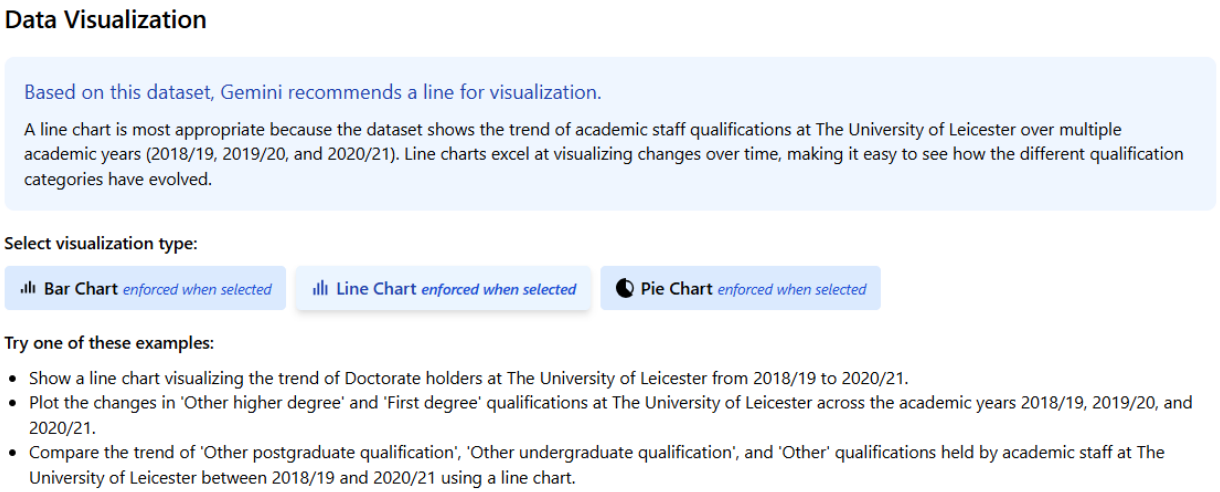


Figure 15: Interactive Data Visualization initially generated by Gemini

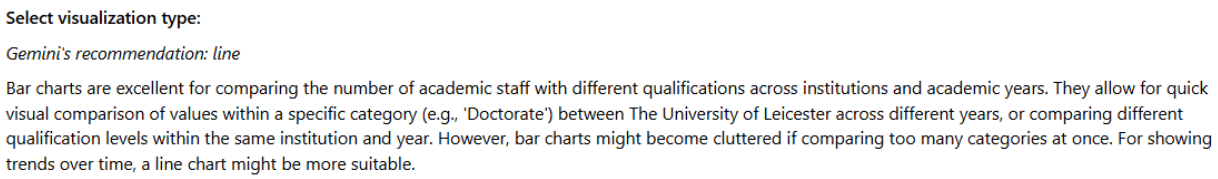


Figure 16: Re-generated recommendation message after different type of chart is selected

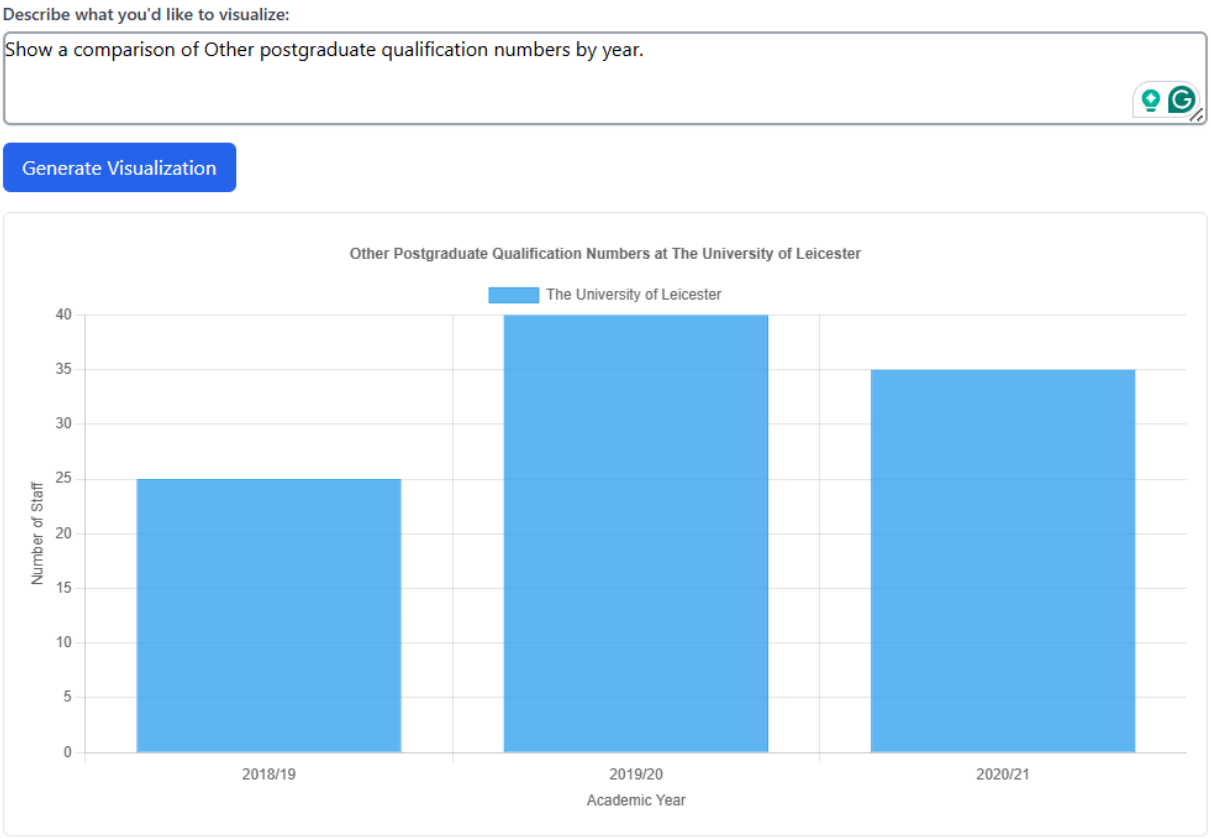


Figure 17: Generated Bar Chart based on specified requirements from selected dataset

Figure 18: Generated Insights of generated bar chart from Figure 17.

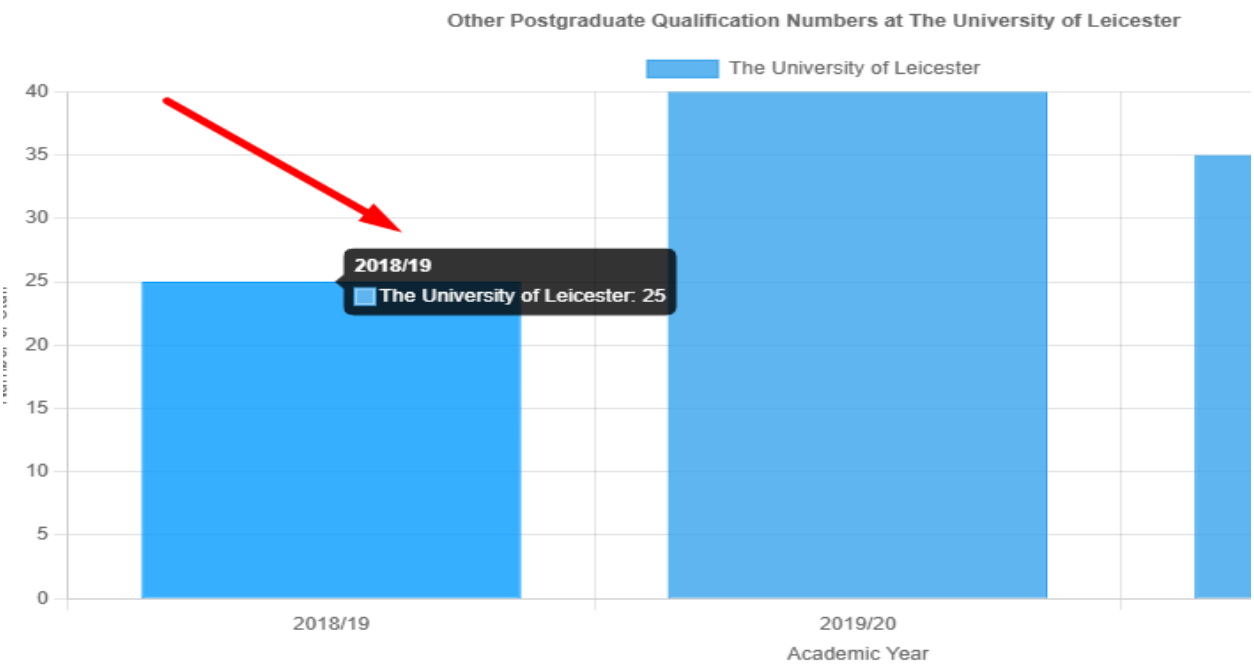


Figure 19: Hovering over data points to see more detailed information

6.3 Providing Feedback on Results

Submitting Feedback on Query and Visualization

Endpoint: */save_feedback* (called in background)

1. Locate the feedback section (see Figure 20)
    - Found at the top of webpage after query is submitted
    - Look for buttons “Not Helpful”, “Helpful” and “Very Helpful”

2. Provide rating and comments (see Figure 21)
    - Select “Not Helpful”, “Helpful” or “Very Helpful” to rate the result
    - Enter detailed comments if appropriate rating is selected in the feedback text area (optional)
    - Example comment: “The visualization clearly showed the trend in postgraduate numbers but only one dataset was returned”

3. Submit the feedback (see Figure 22)
    - Click “Submit Feedback” button
    - A conformation message will appear after that

Figure 20: Feedback Submission Interface

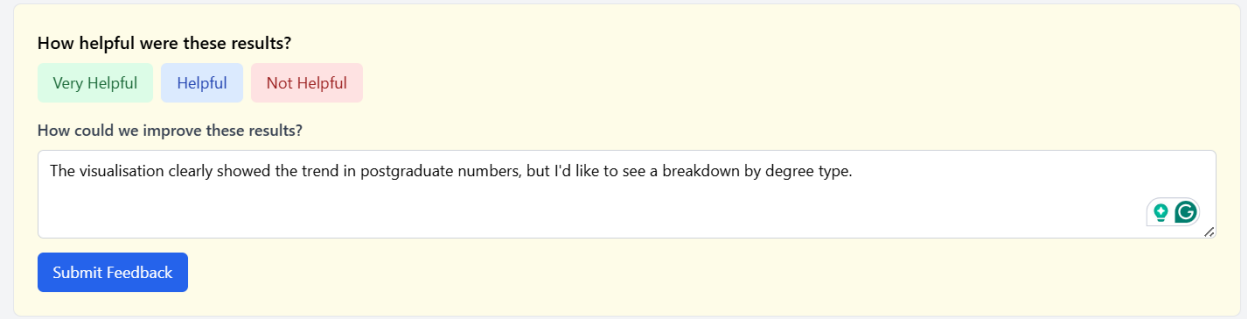


Figure 21: Provided comment for feedback

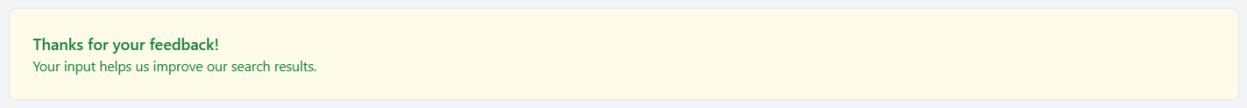


Figure 22: Submitted feedback

6.4 Uploading Custom Data

Navigating to the Data Upload Section

Endpoint: */upload-data*

1. Access the upload page (see Figure 22)
    - Click "Upload Date" in the navigation menu
    - The system will redirect to the data upload interface

2. Understand upload requirements (see Figure 22)
    - View instructions for supported file formats (CSV)
    - See example of correctly formatted files
    - Read about file size limits and columns formatting requirements

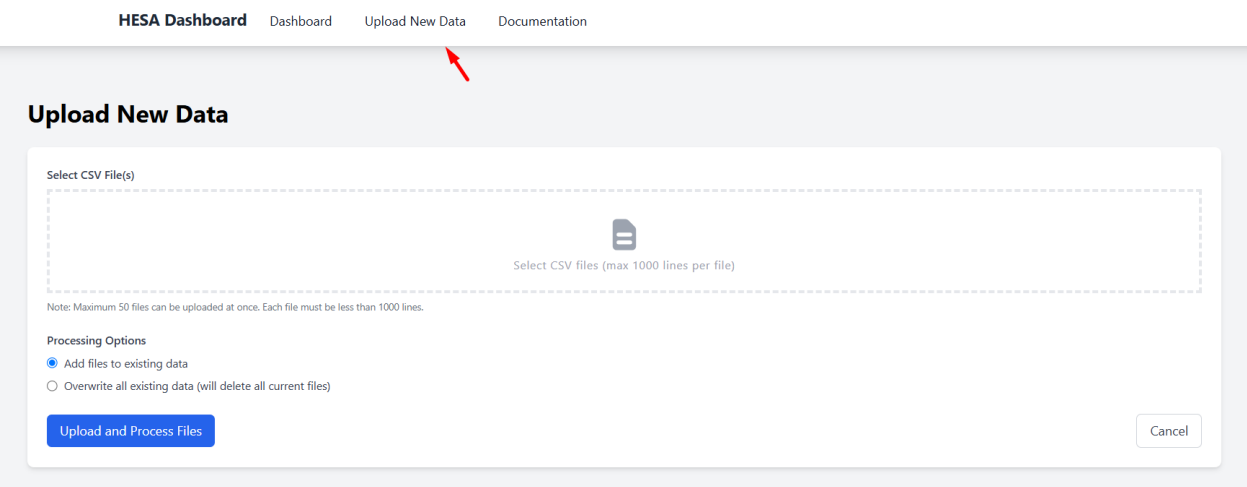


Figure 22: Data Upload Interface

Uploading and Processing Data Files

Endpoint: */process-upload* (called when uploading)

1. Select files to upload (see Figure 23)

- Click “Choose Files” or drag and drop files into the upload area
- Select one or more CSV files with HESA compatible data
- Example raw file: “dt025-table-1 (2).csv”

2. Configure upload options
    - Add files to existing data
    - Overwrite all existing data (it will delete all current files)

3. Initiate upload process (see Figure 24)
    - Click “Upload and Process Files” button to begin processing
    - Progress indicator shows upload and processing status

4. Review processing results (see Figure 25)
    - See confirmation of successful upload
    - View any validation warnings or errors

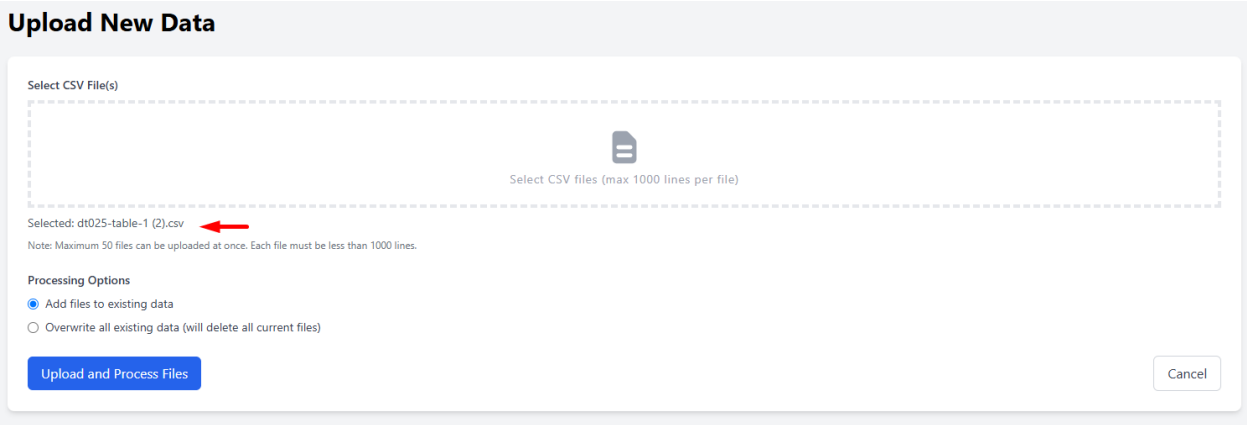


Figure 23: Select files to be uploaded

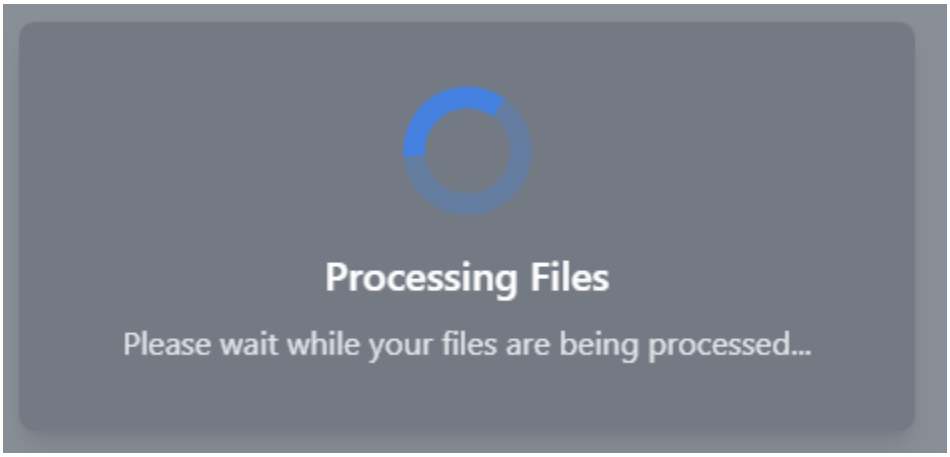


Figure 24: Uploading progress indicators

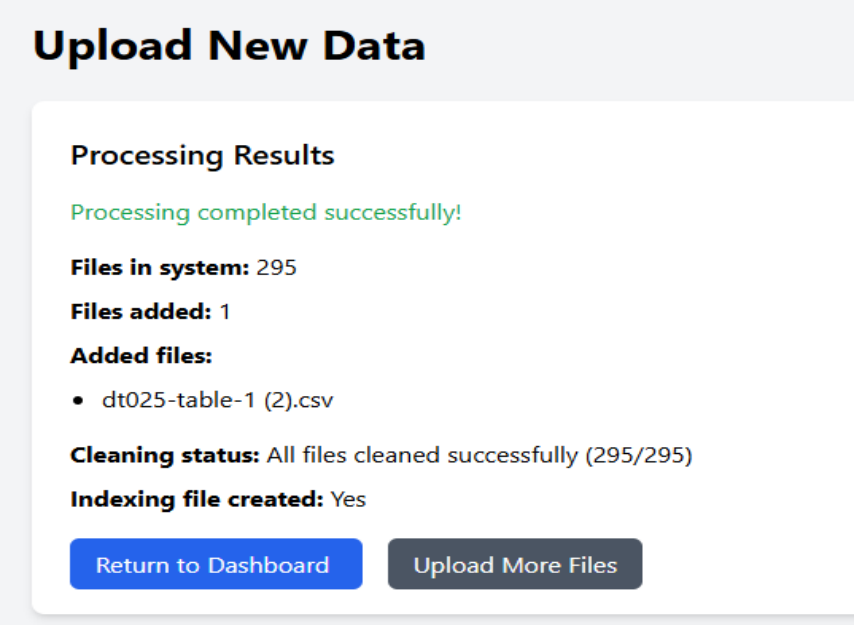


Figure 25: Conformation of successful upload

6.5 Exploring Documentation

Accessing the Documentation Page

Endpoint: /documentation

1. Navigate to documentation (see Figure 26)
    - Click “Documentation” in the navigation menu
    - Documentation page provides guidance on how to use the system

2. Explore available documents sections (see Figure 27 - 33)
    - Introduction to the HESA dashboard
    - Guide to formatting effective queries
    - Explanation of visualization options
    - Data dictionary with columns descriptions
    - Tips and Ticks that can help to simple the use of the system

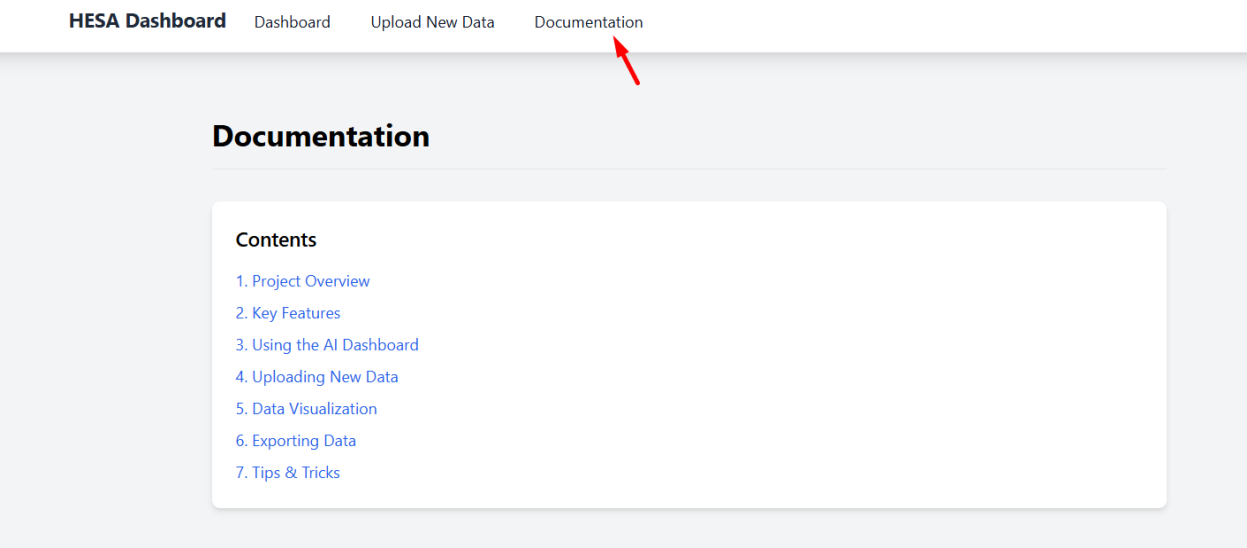


Figure 26: Documentation Page

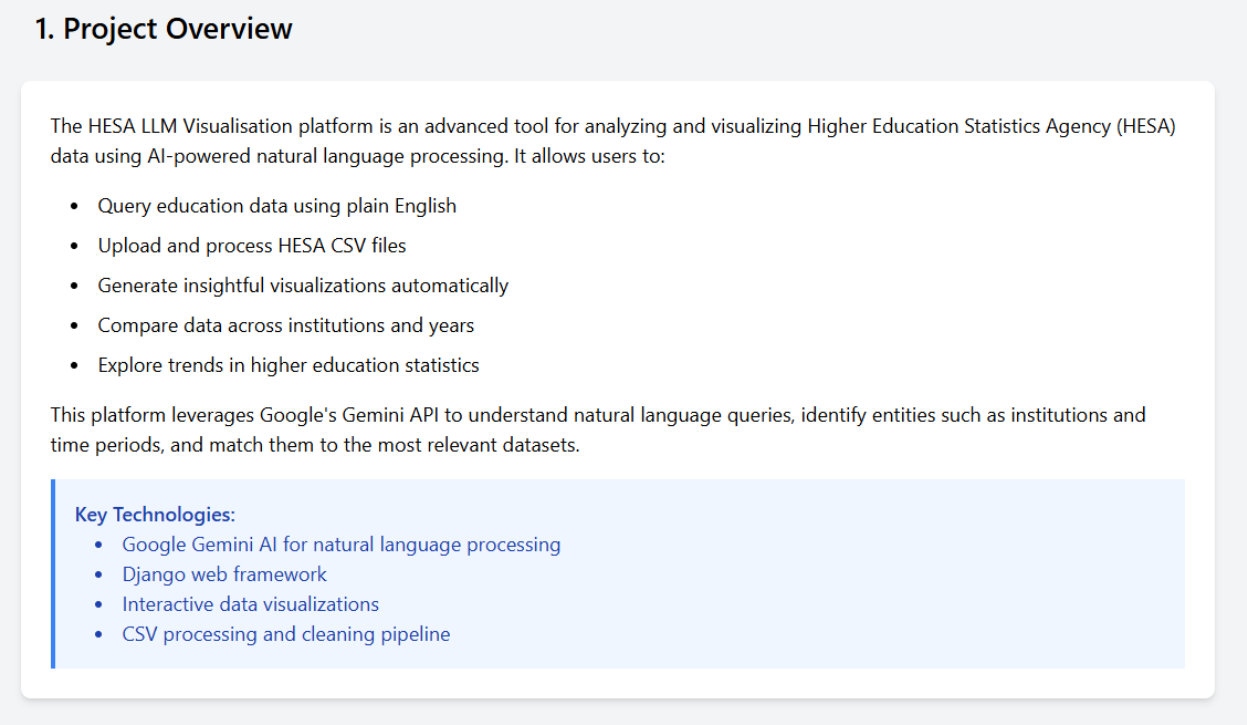


Figure 27: Project Overview Information

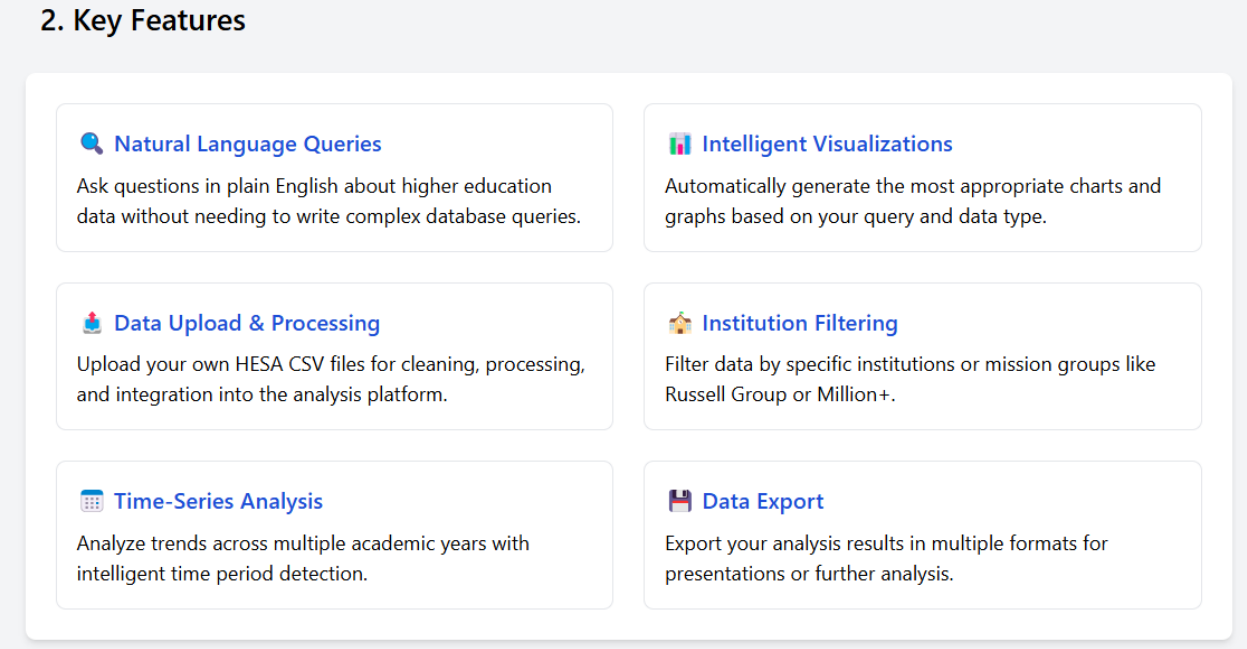


Figure 28: Key Features

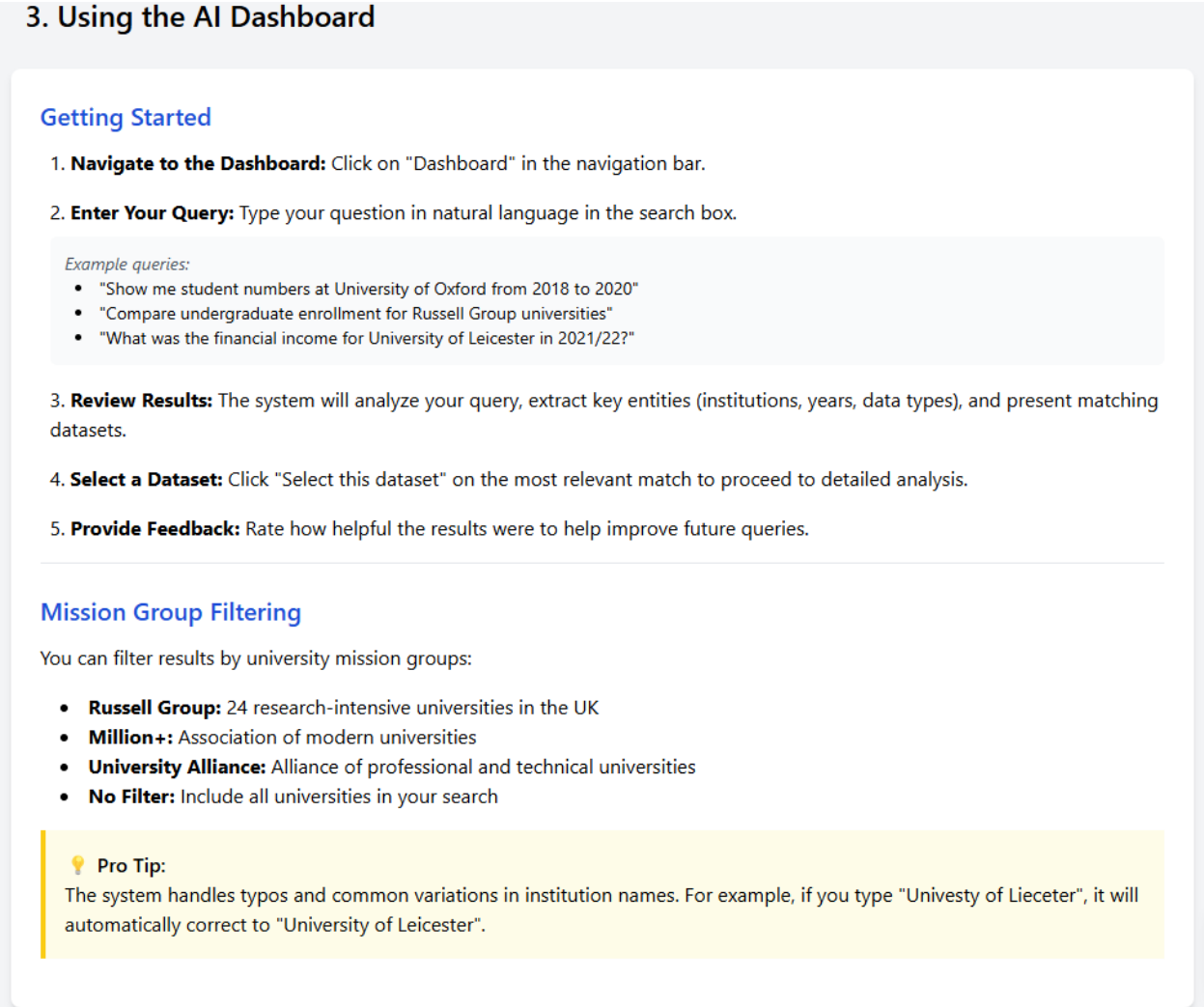


Figure 29: Introduction to project UI

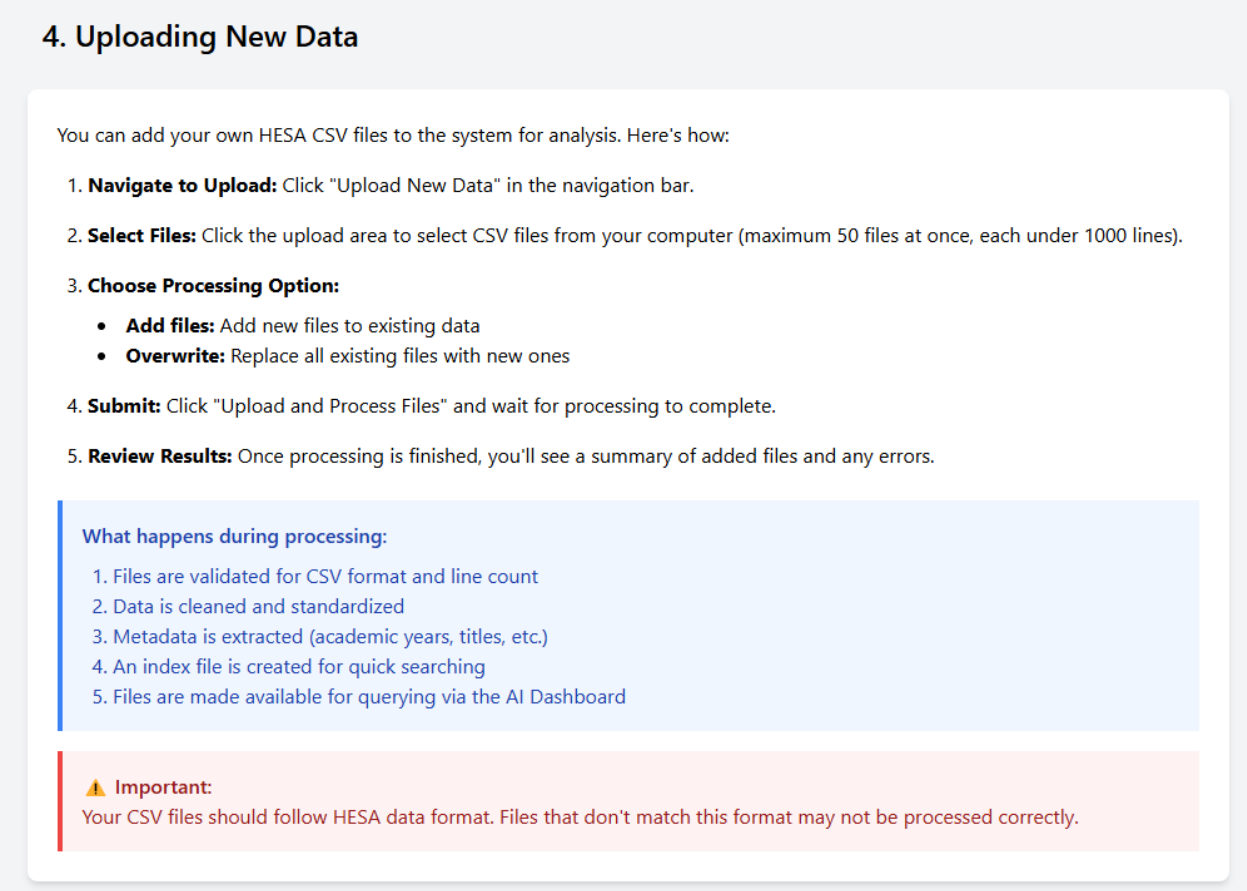


Figure 30: Uploading new data guide

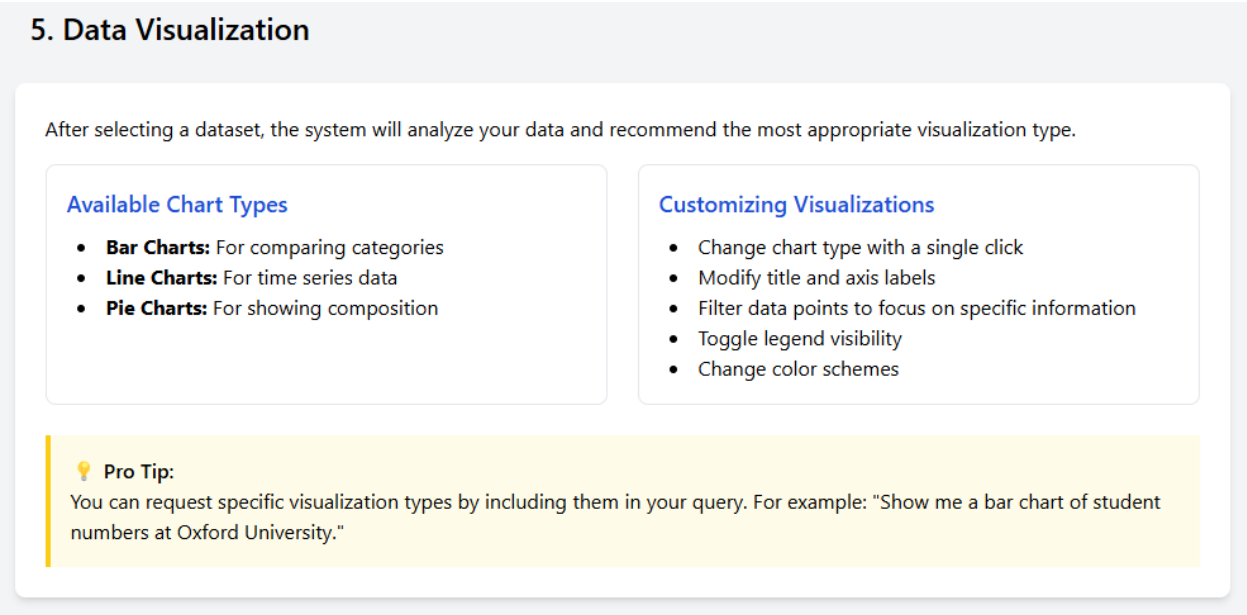


Figure 31: Guide on data visualisation

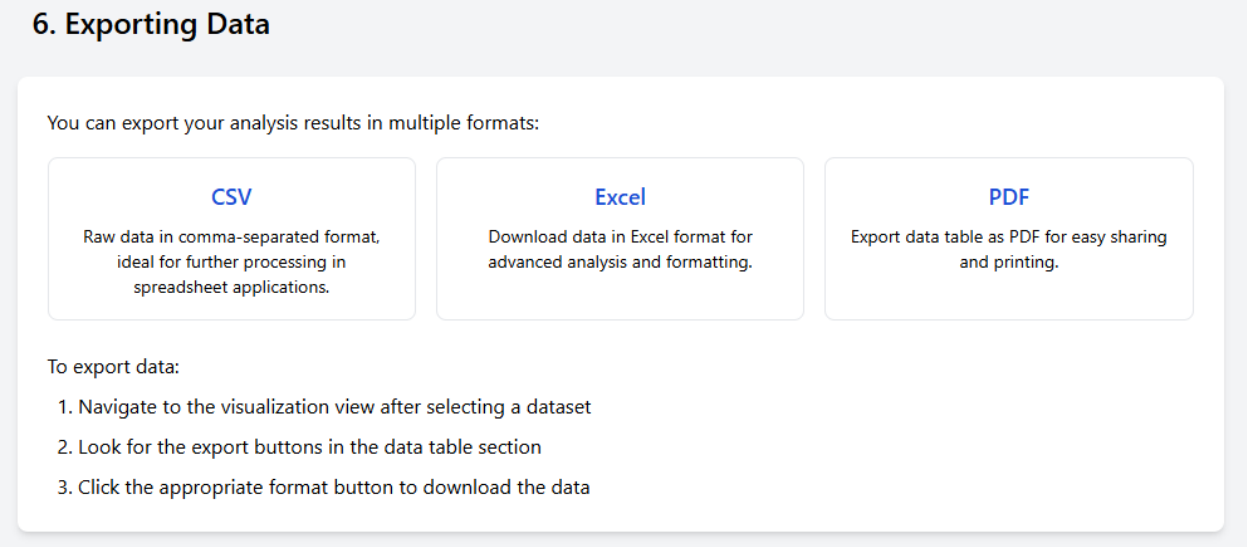


Figure 32: Explanation of exporting data

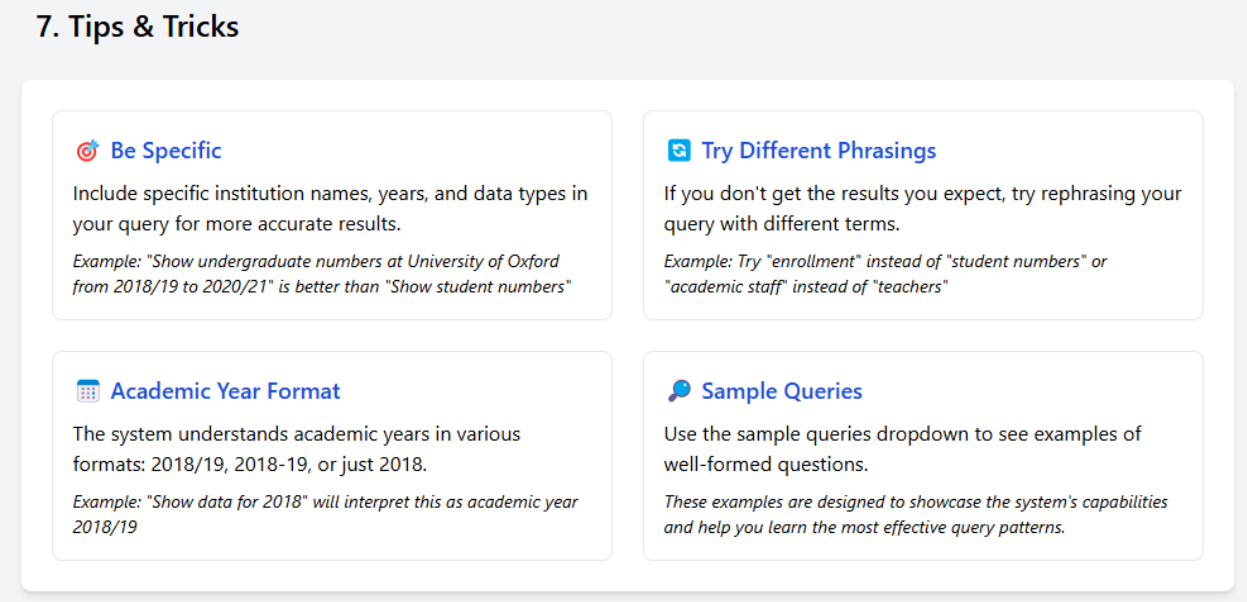


Figure 33: Tips and Tricks about the project

6.6 Complete User Journey Example

From Query to Insight: Full Workflow

This demonstrates a complete user journey through the system

1. Start at dashboard (/ endpoint) (see Figure 34)
    - Enter a query: “How many full-time students lived in university accommodation at University of York in the past 5 years?”
    - Set matches to show to 3
    - Select mission group to equal Russell Group

2. View query analysis (after */process_gemini_query/* is called) (see Figure 35)

- The system extracts institutions: “University of Leicester”, “University of York” and list of 24 institutions from Russell Group
- The system extracts years: “2020-2024” (there is no available data for 2025 academic year yet)
- The system identifies data requested: “full-time students accommodation”

3. Provide feedback (submits to */save_feedback*) (see Figure 40)
    - Rate results as “Helpful”
    - Provide comment: “The visualization clearly shows the trend of students in the provided maintained property”
    - Submit feedback and receive conformation

4. Select a dataset (triggers */ai_dataset_details*) (see Figures 36,37)
    - Choose the most relevant dataset (“Full-time HE student enrolments by HE provider and term-time accommodation”)
    - View detailed data preview filtered to the universities and specified years

5. Generate visualization (uses */visualization_api*) (see Figures 38,39)
    - System recommends: Line chart
    - See the explanation: “A line chart is most appropriate because the dataset includes data across multiple academic years, making it ideal for visualizing trends in student accommodation choices over time for each institution”
    - Enter desired visualization request: “Show the trend of students living in Provider maintained property at Queen's University Belfast and The University of Birmingham from 2020/21 to 2023/24”
    - Generated visualization: A line chart with years on x-axis, number of students on y-axis, for Queen's University Belfast and The University of Birmingham institutions about Provider maintained property

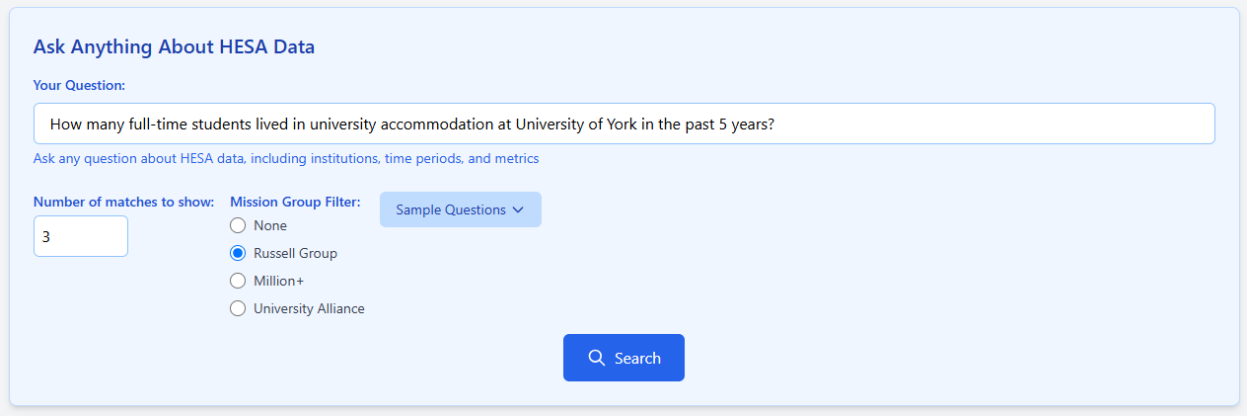


Figure 34: Submitting a query

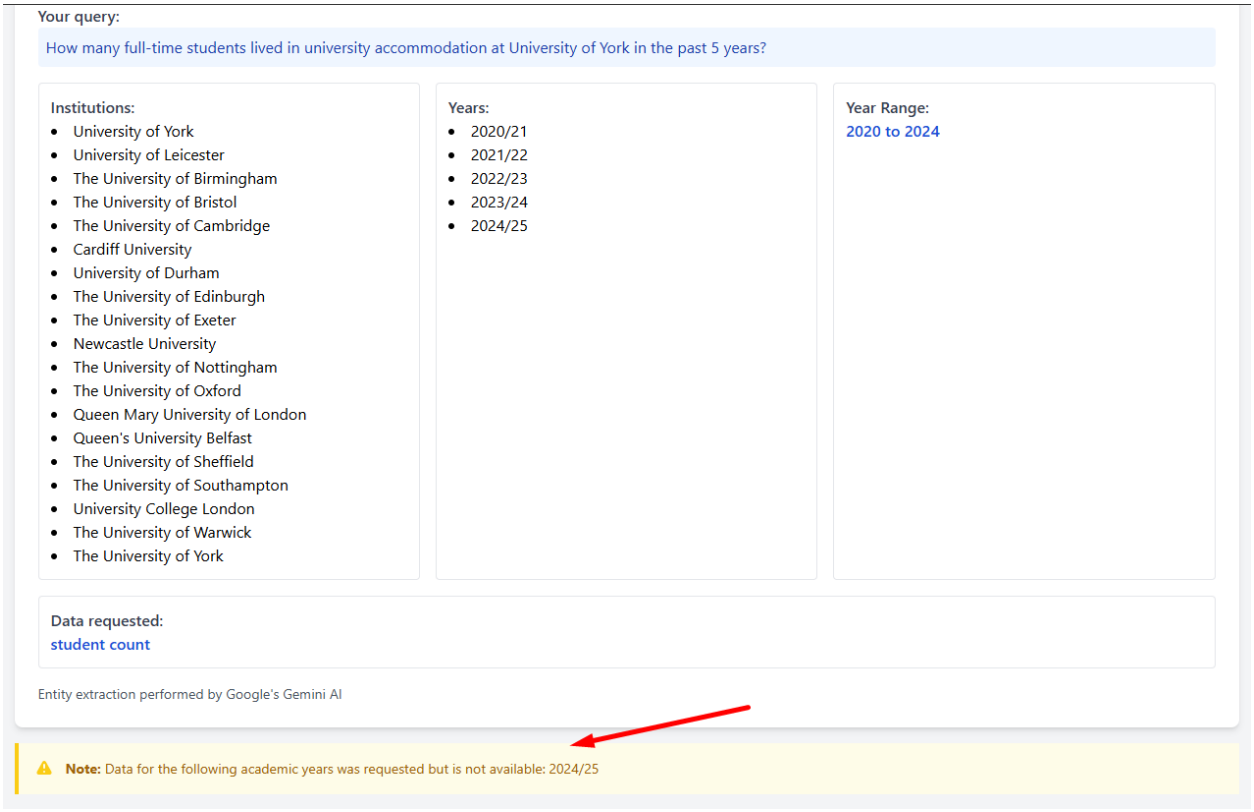


Figure 35: Query parameters extracted by Gemini

Figure 36: Preview of chosen dataset

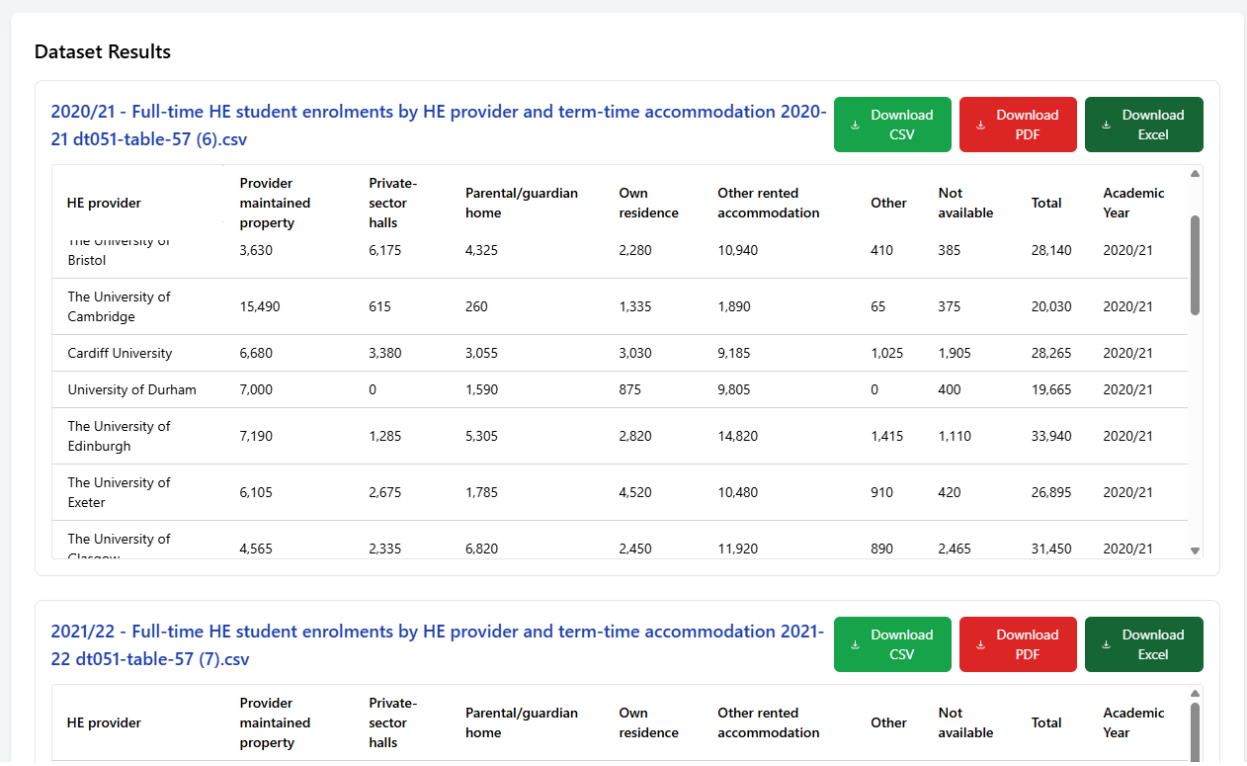


Figure 37: Selected dataset (no longer in preview mode)

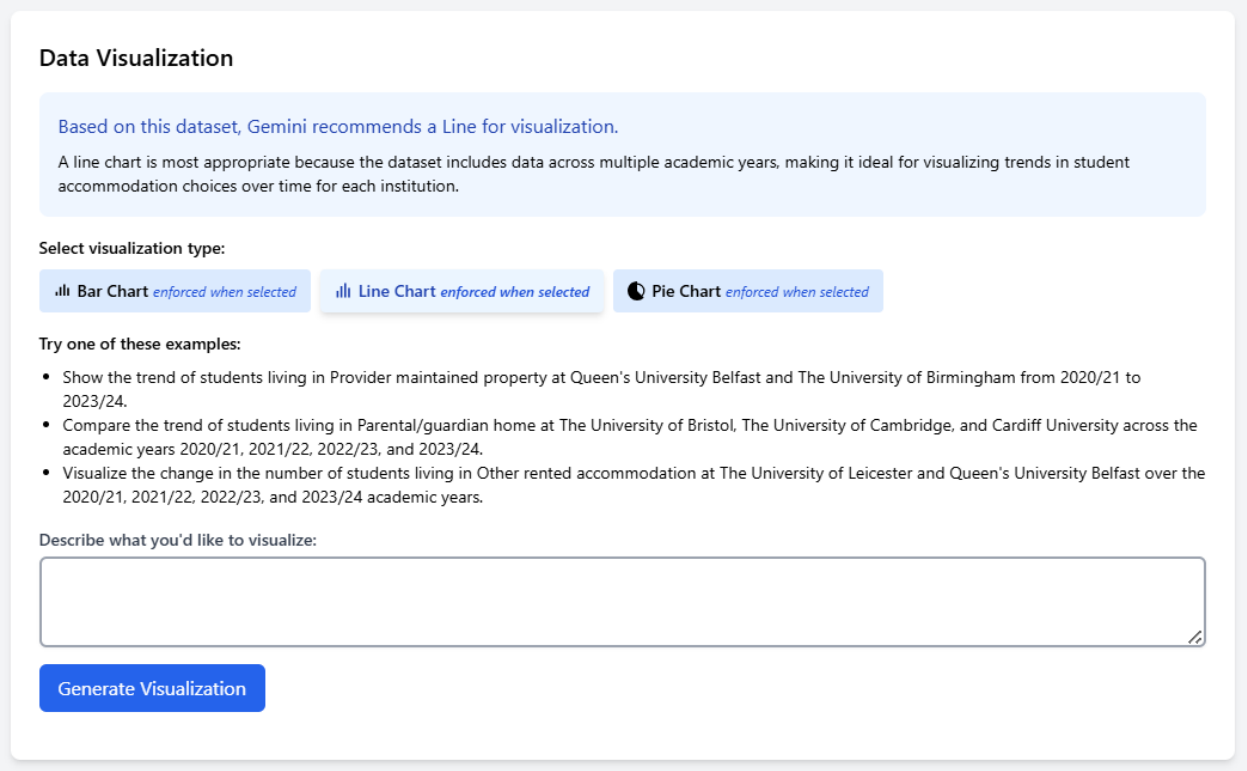


Figure 38: Visualization interface of selected dataset

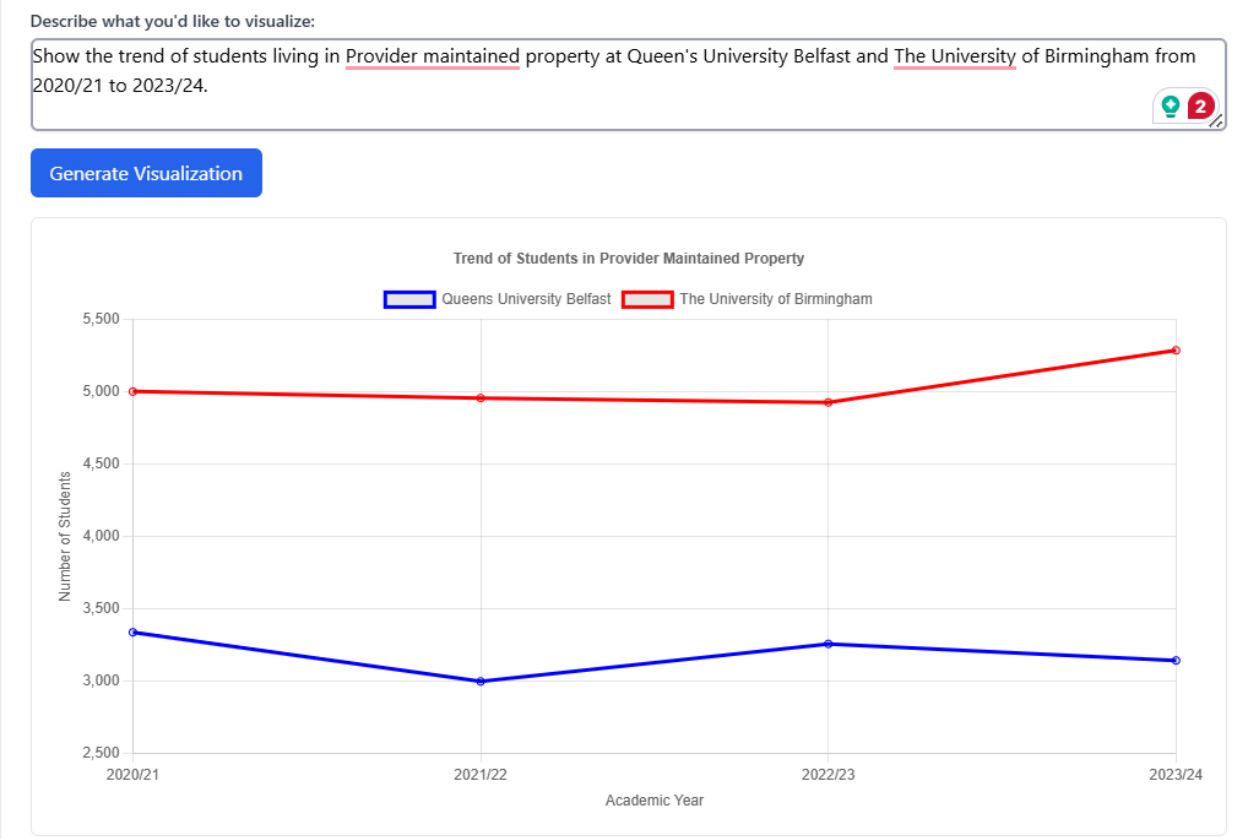


Figure 39: Generated visualisation based on chosen requirements

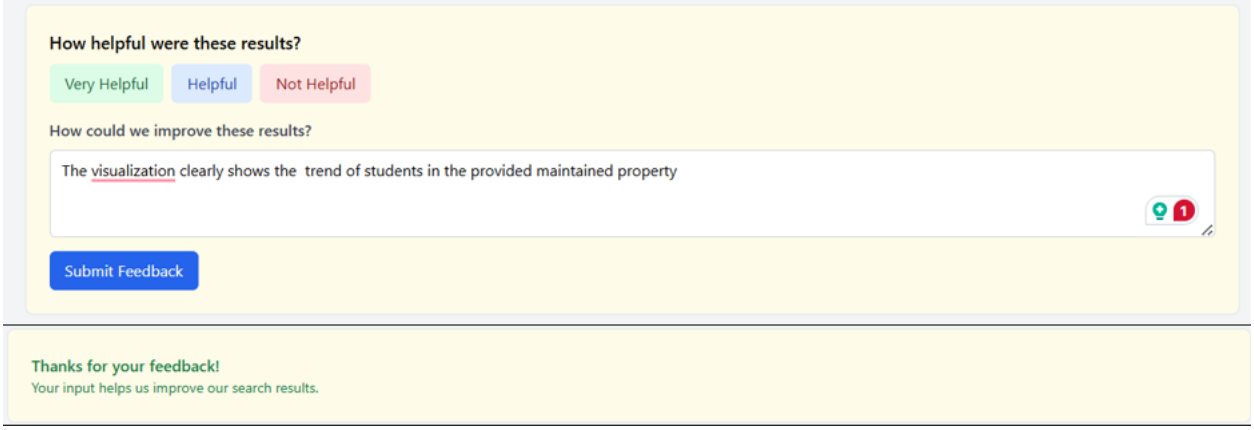


Figure 40: Submitted feedback as Helpful

7. Testing and Evaluation

7.1 Testing Methodologies

Unit Testing

The unit tests covered three main components of the application:

1. Data Processing - CSV data handling and processing
2. LLM Integration - Query parsing and language model interaction

3. Visualization - Chart generation and recommendation

All unit tests successfully executed. (see Figure 41)

CSV Processor (see Appendix B.1)

The CSVProcessor class was tested for its ability to:

- Validate CSV File
  - test_validate_csv_valid_file
  - test_validate_csv_invalid_file
  - test_validate_csv_nonexistent_file
- Find the data start point in HESA files
  - test_find_data_start
- Clean and process CSV data
  - test_clean_csv
- Extract metadata from files
  - test_extract_metadata
- Extract keyword from titles and columns
  - test_extract_keywords_from_title
  - test_extract_keywords_from_columns

Query Parser (see Appendix B.2)

The query parsing functionality was tested for:

- Basic query parsing
  - test_parse_hesa_query_basic
- Local query analysis
  - test_local_analyze_query
- Handling queries with multiple institutions
  - test_query_with_multiple_institutions
- Handling year ranges
  - test_query_with_year_range
- Handling academic year formats
  - test_query_with_academic_year

Chart Generation (see Appendix B.3)

The visualization component was tested for:

- Chart type recommendations
  - test_get_chart_recommendation

- Visualization generation
    - test_generate_visualization

- Changing chart type
    - test_change_chart_type

- Explaining chart type selection
    - test_chart_type_explanation

```
tests/unit/data_processing/test_csv_processor.py::TestCSVProcessor::test_validate_csv_valid_file
PASSED [  5%]
tests/unit/data_processing/test_csv_processor.py::TestCSVProcessor::test_validate_csv_invalid_fil
e PASSED [ 11%]
tests/unit/data_processing/test_csv_processor.py::TestCSVProcessor::test_validate_csv_nonexistent
_file PASSED [ 17%]
tests/unit/data_processing/test_csv_processor.py::TestCSVProcessor::test_find_data_start PASSED [
 23%]
tests/unit/data_processing/test_csv_processor.py::TestCSVProcessor::test_clean_csv PASSED [ 29%]
tests/unit/data_processing/test_csv_processor.py::TestCSVProcessor::test_extract_metadata PASSED
[ 35%]
tests/unit/data_processing/test_csv_processor.py::TestCSVProcessor::test_extract_keywords_from_ti
tle PASSED [ 41%]
tests/unit/data_processing/test_csv_processor.py::TestCSVProcessor::test_extract_keywords_from_co
lumns PASSED [ 47%]
tests/unit/llm/test_query_parser.py::TestQueryParser::test_parse_hesa_query_basic PASSED [ 52%]
tests/unit/llm/test_query_parser.py::TestQueryParser::test_local_analyze_query PASSED [ 58%]
tests/unit/llm/test_query_parser.py::TestQueryParser::test_query_with_multiple_institutions PASSE
D [ 64%]
tests/unit/llm/test_query_parser.py::TestQueryParser::test_query_with_year_range PASSED [ 70%]
tests/unit/llm/test_query_parser.py::TestQueryParser::test_query_with_academic_year PASSED [ 76%]
tests/unit/visualization/test_chart_generation.py::TestChartGeneration::test_get_chart_recommenda
tion PASSED [ 82%]
tests/unit/visualization/test_chart_generation.py::TestChartGeneration::test_generate_visualizati
on PASSED [ 88%]
tests/unit/visualization/test_chart_generation.py::TestChartGeneration::teseeeest_chesteeeesestes
est_change_chart_type PASSED [ 94%]
tests/unit/visualization/test_chart_generation.py::TestChartGeneration::test_chart_type_explanati
on PASSED [100%]

========================== 17 passed in 2.83s ==========================
```

Figure 41: Passed Unit Testing 100%

Integration Testing

I created integration tests that verify the end-to-end flow of data through the HESA LLM Visualization system. It focuses on how different components work together. The integration tests cover:

1. Query to Visualization Flow – Testing the complete pipeline from user query to final visualization
2. Web Interface Integration – Testing how the API endpoints integrate with the code processing functions

Key Flows Tested

1. Query Processing Flow
    - Natural language query (NLQ) is parsed to extract institutions, years, and data types
    - The extracted information is used to find matching datasets
    - Test verifies that a query like “How many students were at University of Cambridge in 2020?” correctly extracts "University of Cambridge" and "2020"

2. Data Retrieval Flow
    - Tests that query parameters can be used to find relevant CSV files
    - Verifies metadata extraction from files works correctly
    - Checks that data can be loaded from the matched files

3. Visualization Generation Flow
    - Tests the pipeline from data to chart recommendation and generation
    - Verifies that appropriate chart types are recommended based on data characteristics
    - Ensures visualization configurations include proper chart data, options, and insights

4. End-to-End Query to Chart Flow
    - Tests the complete flow from query to visualization
    - Validates that a query like “Show me student enrolments by university in 2022/23” results in appropriate data selection and visualization

All integration tests were executed successful. (see Figure 41)

```
Running integration tests...

============================= test session starts ==============================
platform win32 -- Python 3.12.4, pytest-8.3.5, pluggy-1.5.0 -- C:\Users\vladi\Desktop\vr112\hesa-llm-visualisation\venv\
Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\vladi\Desktop\vr112\hesa-llm-visualisation
plugins: anyio-4.9.0, cov-6.1.1
collected 7 items

tests/integration/test_query_to_visualization_flow.py::TestQueryToVisualizationFlow::test_query_parse_to_data_retrieval
PASSED [ 14%]
tests/integration/test_query_to_visualization_flow.py::TestQueryToVisualizationFlow::test_data_to_visualization_flow PAS
SED [ 28%]
tests/integration/test_query_to_visualization_flow.py::TestQueryToVisualizationFlow::test_end_to_end_query_processing PA
SSED [ 42%]
tests/integration/test_web_interface.py::TestWebInterface::test_process_query_endpoint
 PASSED [ 57%]
tests/integration/test_web_interface.py::TestWebInterface::test_visualization_endpoint
 PASSED [ 71%]
tests/integration/test_web_interface.py::TestWebInterface::test_chart_recommendation_endpoint PASSED [ 85%]
oint PASSED [100%]

============================== 7 passed in 3.12s ===============================

Integration tests completed!
```

Figure 42: Passed Integration Testing 100%

Query to Visualization Flow (see Appendix C.1)

Location: test_query_to_visualization_flow.py

This file contains test that verify the complete data pipeline from a NLQ to a visualization output. These test focus on:

- Data Flow Integration: How data moves between different components of the system
- End-to-End Processing: Testing the complete journey from user query to final visualization
- Component Interaction: Verifying that data processing, query parsing and visualization generation work together

Key tests include:

- Testing how NLQs are parsed to find relevant datasets
    - test_query_parse_to_data_retrieva
- Testing transforming data into visualisations with appropriate chart types
    - test_data_to_visualization_flow
- Testing the entire pipeline from query to chart generation
    - test_end_to_end_query_processing

They simulate a user entering a query like "Show me student enrolments at University of Cambridge in 2020" and test that the system correctly:

1. Extracts entities (universities, years)
2. Finds matching datasets
3. Processes the data
4. Generates appropriate visualizations

Mock AI Client Integration Tests (see Appendix C.2)

Location: test_web_interface.py

The tests below focus on the integration between the application and the AI client that handles:

- Chart recommendations
- Visualization generation
- Chart type switching

Key tests include:

- Verifying that appropriate chart types are recommended based on selected data
    - test_chart_recommendation
- Testing the creation of complete visualization configurations
    - test_visualization_generation
- Testing the ability to change chart types for the same data
    - test_chart_type_change

It ensures that the AI client (MockAIClient) correctly integrates with the application by:

1. Providing sensible chart recommendations based on data structure
2. Generating complete chart configurations with all required parameters

3. Supporting different visualization types for the same dataset

7.2 Performance Metrics and Results

Average response time (ART)

- Mean time from query submission to results displayed
- Target:
  - < 3 seconds for easy query
  - < 5 seconds for medium query
  - < 8 seconds for hard query

Cache hit response time (CHRT)

- Response time when query is results are retrieved from cached
- Target:
  - < 0.5 seconds

Query Response Time

Complexity	ID	Query	Response Time (seconds)	Cached Response Time (seconds)
Easy	1	How many postgraduates are in University of Cambridge in 2017? (2 matches, no missing group filtering)	6.21	0.06
	2	What was the research income starting in 2017? (2 matches, no missing group filtering)	5.48	0.05
	3	Show carbon emissions for Coventry University in 2021 (1 matches, no missing group filtering)	4.29	0.11
Medium	4	How many full-time students lived in university accommodation at University of York from 2016 to 2018? (4 matches, no missing group filtering)	8.67	0.1
	5	Research funding for University of Manchester from 2017 to 2019? (7 matches, no missing group filtering)	10.31	0.08
	6	Graduation rates at University of London institutions for the past 5 years (5 matches, no missing group filtering)	58.3	0.67
Hard	7	Compare teaching and salary in University of Bath (3 matches, filtered by Million+)	23.69	0.21

	8	What's the trend in total expenditure for all universities in the past 10 years? (matches 5, filtered by Russell Group)	30.81	0.5
	9	What are the classifications in the University of Bristol from 2014 to 2025	25.06	0.24

Table 7: Raw data from JSON file what stores information about data processing time (see Appendix M.2)

Explanation if query complexity:

- Easy
  - Single year
  - Single institution or no specific institutions
  - No mission group filtering
  - Number of matches to show <= 3

- Medium
  - Multiple years
  - Multiple institutions
  - No mission group filtering
  - Number of matches to show > 3

- Hard
  - Any years (when the year is omitted in the submitted query the system searches for any year)
  - Multiple institutions
  - Applied mission group filtering
  - Number of matches to show > 3

After conducting plenty of testing you can clearly see that caching is a must use technique. On average, the caching methods increase the processing by 16.73 times. The code that implements query timing stored received data in JSON format automatically (see Figure 52, Appendix M.1)

Note:
9.2 seconds (Average Initial Response Time) / 0.55 seconds (Average Cached Response Time) ≈ 16.73

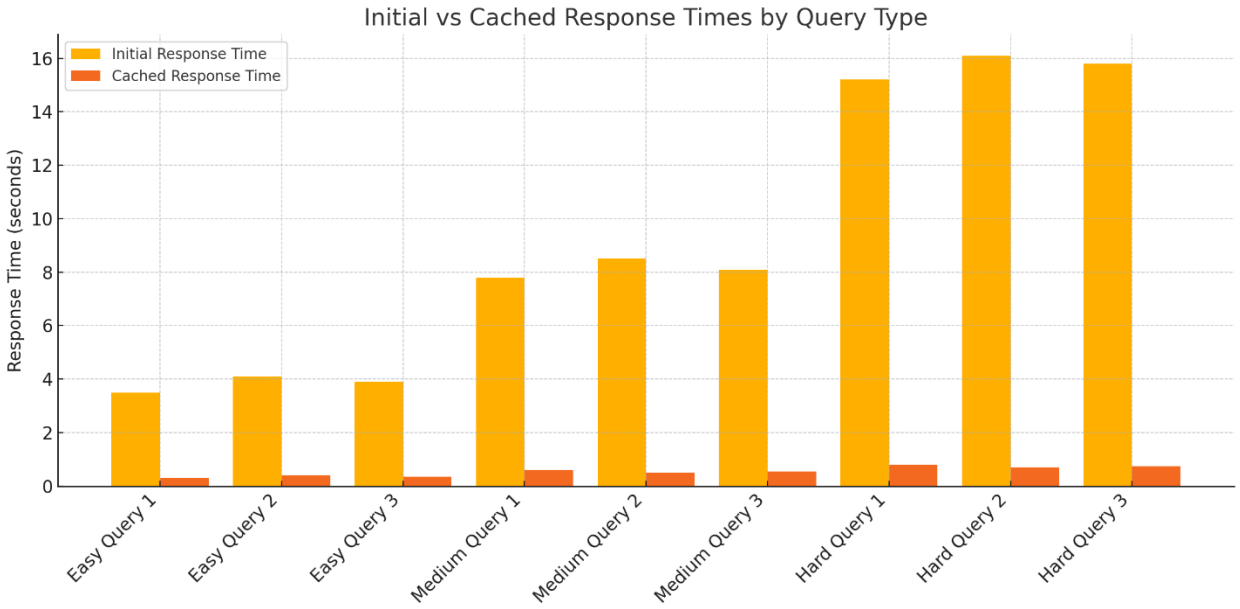


Figure 52: Comparison of Cached and Non-cached approach.

Accuracy of Query Interpretation

The accuracy measurement is calculated based on my own analyses of submitted query and received results.

Relevance scoring (0 – 3 scale):

- 3 = Excellent: Results perfect match the query intent and showing data that I was expected to see
- 2 = Good: Results mostly match the query intent with minor difference
- 1 = Partial: Results are related to the query but missing key aspects or has irrelevant information
- 0 = Poor: Results do not address the query intent or completely off topic.

ID	Query	Score	Figure Number
1	How many postgraduates are in University of Cambridge in 2017?	3	43
2	What was the research income starting in 2017?	3	44
3	Show carbon emissions for Coventry University ending in 2021	3	45
4	How many full-time students lived in university accommodation at University of York from 2016 to 2018?	3	46
5	Research funding for University of Manchester from 2017 to 2019?	2	47
6	Graduation rates at University of London institutions for the past 5 years	1	48
7	Compare teaching and salary in University of Bath	1	49
8	What's the trend in total expenditure for all universities in the past 10 years?	3	50

9	What are the classification in the University of Bristol from 2014 to 2025	2	51

Table 8: Accuracy relevance score on scale 0 to 3

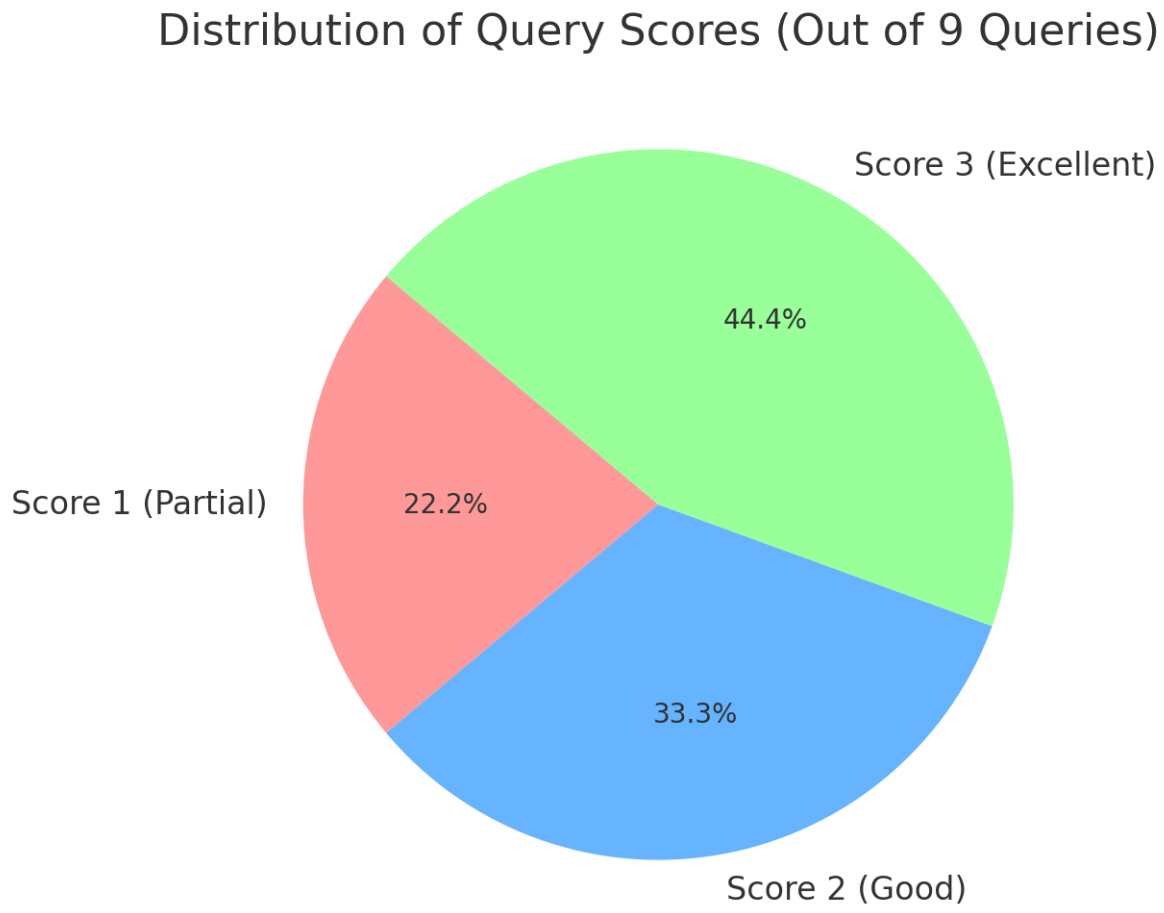


Figure 53: Pie chart of data processing accuracy

The dataset matching works, but there is still room for the improvements. Even though only 44% of the 9 tested queries were evaluated as excellent, this is a solid foundation that can improve to reach 100%. (see Figure 53)

Factors affecting the accuracy of results:

- Even though the system uses the submitted query intent to find the data, there are limits on how deep the Gemini understands the intent. This could be specified in more details and advanced so get better results.
- HESA does provide a lot of data but not everything, so there is chance that the data user is looking for just does not exist, so user end up getting most closely related data.
- The system is capable of understanding typos and closely related words, but this aspect can affect how well Gemini will interpret and process the query

2017/18 - HE qualifiers by HE provider and level of qualification obtained 2017-18 dt051-table-16 (3).csv

He Provider	Doctorate Research	Other Postgraduate Research	Master's Taught	Postgraduate Certificate In Education	Other Postgraduate Taught	Total Postgraduate	First Degree	Foundation Degree	Hnc/hnd	Professio Graduate Certificat In Educatio
The University of Cambridge	1,115	940	1,860	265	220	4,400	3,275	0	0	0
The Cambridge Theological	0	0	10	0	10	20	0	0	0	0

Showing 3 of 4 matching rows

Figure 43: Found Data for Easy Query 1

2017/18 - Finances and people 2017-18 dt042-table-5 (2).csv

He Provider	Total Income (£)	Teaching Income (£)	Research Income (£)	Other Non-residential Income (£)	Non-residential Income Total (£)	Total Expenditure (£)	Academic Year
HE provider	Total income (£)	Teaching income (£)	Research income (£)	Other non-residential income (£)	Non-residential income total (£)	Total expenditure (£)	2017/18
The University of Aberdeen	219,471,000	107,977,000	78,696,000	21,315,000	207,988,000	226,949,000	2017/18
Abertay University	33,558,000	27,247,000	2,608,000	2,624,000	32,479,000	33,331,000	2017/18

Showing 3 of 3 matching rows

Figure 44: Found Data for Easy Query 2

1. Emissions and waste

Match: 0.90 (90%)

Show Details

Available Files:

2021/22 - Emissions and waste 2021-22 dt042-table-3 (6).csv

He Provider	Total Scope 1 And 2 Carbon Emissions (kg Co2e)	Total Volume Of Wastewater (m3)	Total Scope 3 Carbon Emissions From Waste (tonnes Co2e)	Total Scope 3 Carbon Emissions From Water Supply (tonnes Co2e)	Total Scope 3 Carbon Emissions From Wastewater Treatment (tonnes Co2e)	Total Waste Mass (tonnes)	Academic Year
Coventry University	11,182,244.714	215,790.980	28.533	32.153	59.304	2,651.015	2021/22
The University of Leicester	15,535,379.297	228,685.000	64.121	34.074	62.67	1,093.041	2021/22

Figure 45: Found Data for Easy Query 3

2016/17 - Full-time HE student enrolments by HE provider and term-time accommodation 2016-17 dt051-table-57 (2).csv

He Provider	Provider Maintained Property	Private-sector Halls	Parental/guardian Home	Own Residence	Other Rented Accommodation	Other	Not Available	Total	Academic Year
The University of Leicester	3,425	725	885	2,135	6,310	540	605	14,635	2016/17
York St John University	1,565	0	845	845	1,870	180	50	5,355	2016/17
The University of York	5,585	815	495	4,225	4,280	180	700	16,285	2016/17

2017/18 - Full-time HE student enrolments by HE provider and term-time accommodation 2017-18 dt051-table-57 (3).csv

He Provider	Provider Maintained Property	Private-sector Halls	Parental/guardian Home	Own Residence	Other Rented Accommodation	Other	Not Available	Total	Academic Year
The University of Leicester	3,560	615	1,060	1,680	6,945	450	1,110	15,420	2017/18
York St John University	2,060	0	810	780	1,640	140	300	5,730	2017/18
The University of York	5,360	1,655	565	4,250	4,690	335	365	17,220	2017/18

2018/19 - Full-time HE student enrolments by HE provider and term-time accommodation 2018-19 dt051-table-57 (4).csv

Figure 46: Found Data for Medium Query 1

Available Files:

2017/18 - Finances and people 2017-18 dt042-table-5 (2).csv

He Provider	Total Income (£)	Teaching Income (£)	Research Income (£)	Other Non-residential Income (£)	Non-residential Income Total (£)	Total Expenditure (£)	Academic Year
The University of Bolton	53,127,000	48,266,000	850,000	2,496,000	51,612,000	54,179,000	2017/18
The University of Bristol	657,853,000	307,279,000	215,490,000	92,189,000	614,958,000	650,822,000	2017/18
The University of Central Lancashire	226,888,000	182,533,000	11,078,000	24,144,000	217,755,000	228,264,000	2017/18

Showing 3 of 20 matching rows

2018/19 - Finances and people 2018-19 dt042-table-5 (3).csv

He Provider	Total Income (£)	Teaching Income (£)	Research Income (£)	Other Non-residential Income (£)	Non-residential Income Total (£)	Total Expenditure (£)	Academic Year
The University of Bolton	86,068,000	78,291,000	920,000	5,183,000	84,394,000	87,358,000	2018/19
The University of Bristol	706,891,000	337,020,000	219,595,000	100,355,000	656,970,000	776,290,000	2018/19
The University of Central Lancashire	234,838,000	187,822,000	11,426,000	26,591,000	225,839,000	249,822,000	2018/19

Showing 3 of 20 matching rows

Figure 47: Found Data for Medium Query 2

Central Film School London	0	0	0	0	0	0	0	5	2020/21
City, University of London	1,050	800	320	205	60	40	45	2,515	2020/21

Showing 3 of 30 matching rows

2021/22 - HE academic staff by HE provider and highest qualification held 2021-22 dt025-table-8 (7).csv

He Provider	Doctorate	Other Higher Degree	Other Postgraduate Qualification	First Degree	Other Undergraduate Qualification	Other	Unknown	Total Academic Staff	Academic Year
Brunel University London	880	270	40	90	20	10	10	1,315	2021/22
Central Film School London	0	0	0	0	0	0	0	5	2021/22
City, University of London	835	540	185	120	30	20	20	1,750	2021/22

Showing 3 of 30 matching rows

2022/23 - HE academic staff by HE provider and highest qualification held 2022-23 dt025-table-8 (8).csv

He Provider	Doctorate	Other Higher Degree	Other Postgraduate Qualification	First Degree	Other Undergraduate Qualification	Other	Unknown	Total Academic Staff	Academic Year

Figure 48: Found Data for Medium Query 3

1. Expenditure - breakdown by activity and HESA cost centre

Match: 0.70 (70%)

Show Details

Available Files:

2015/16 - Expenditure - breakdown by activity and HESA cost centre 2015-16 dt031-table-8.csv

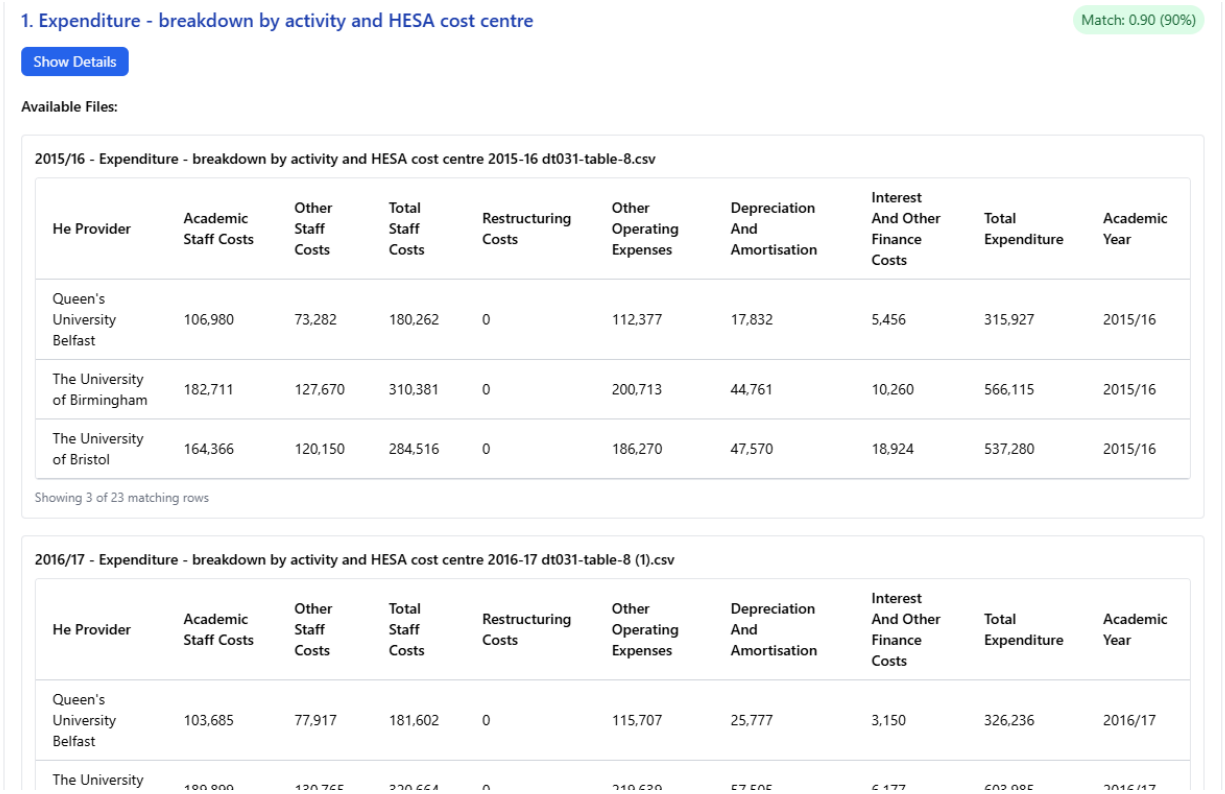
He Provider	Academic Staff Costs	Other Staff Costs	Total Staff Costs	Restructuring Costs	Other Operating Expenses	Depreciation And Amortisation	Interest And Other Finance Costs	Total Expenditure	Academic Year
Bath Spa University	18,770	16,556	35,326	0	24,742	4,489	1,930	66,487	2015/16
The University of Bath	68,734	62,315	131,049	0	86,582	22,136	9,373	249,140	2015/16
The University of Bolton	16,657	10,548	27,205	97	15,792	2,541	814	46,449	2015/16

Showing 3 of 20 matching rows

2016/17 - Expenditure - breakdown by activity and HESA cost centre 2016-17 dt031-table-8 (1).csv

He Provider	Academic Staff Costs	Other Staff Costs	Total Staff Costs	Restructuring Costs	Other Operating Expenses	Depreciation And Amortisation	Interest And Other Finance Costs	Total Expenditure	Academic Year

Figure 49: Found Data for Hard Query 1



1. Expenditure - breakdown by activity and HESA cost centre

Match: 0.90 (90%)

Show Details

Available Files:

2015/16 - Expenditure - breakdown by activity and HESA cost centre 2015-16 dt031-table-8.csv

He Provider	Academic Staff Costs	Other Staff Costs	Total Staff Costs	Restructuring Costs	Other Operating Expenses	Depreciation And Amortisation	Interest And Other Finance Costs	Total Expenditure	Academic Year
Queen's University Belfast	106,980	73,282	180,262	0	112,377	17,832	5,456	315,927	2015/16
The University of Birmingham	182,711	127,670	310,381	0	200,713	44,761	10,260	566,115	2015/16
The University of Bristol	164,366	120,150	284,516	0	186,270	47,570	18,924	537,280	2015/16

Showing 3 of 23 matching rows

2016/17 - Expenditure - breakdown by activity and HESA cost centre 2016-17 dt031-table-8 (1).csv

He Provider	Academic Staff Costs	Other Staff Costs	Total Staff Costs	Restructuring Costs	Other Operating Expenses	Depreciation And Amortisation	Interest And Other Finance Costs	Total Expenditure	Academic Year
Queen's University Belfast	103,685	77,917	181,602	0	115,707	25,777	3,150	326,236	2016/17
The University	189,899	130,765	320,664	0	219,639	57,505	6,177	603,985	2016/17

Figure 50: Found Data for Hard Query 1

1. HE staff by HE provider and activity standard occupational classification

Match: 0.80 (80%)

Show Details

Available Files:

2014/15 - HE staff by HE provider and activity standard occupational classification 2014-15 dt025-table-1.csv

He Provider	Managers, Directors And Senior Officials	Professional Occupations	Associate Professional Occupations	Clerical And Manual Occupations	Total Academic Staff	Administrative And Secretarial Occupations	Skilled Trades Occupations	Caring, Leisure And Other Service Occupations	Sales And Customer Service Occupations
Anglia Ruskin University	5	760	15	0	780	65	225	265	450
Birmingham City University	5	1,565	5	0	1,580	105	240	205	460

Showing 3 of 18 matching rows

2015/16 - HE staff by HE provider and activity standard occupational classification 2015-16 dt025-table-1 (1).csv

He Provider	Managers, Directors And Senior Officials	Professional Occupations	Associate Professional Occupations	Clerical And Manual Occupations	Total Academic Staff	Administrative And Secretarial Occupations	Skilled Trades Occupations	Caring, Leisure And Other Service Occupations	Sales And Customer Service Occupations

Figure 51: Found Data for Hard Query 3

8. Discussion

8.1 Analysis of Findings

The current system successfully addresses the problem statement and met its aims. The main strength is its capabilities to perform all the core functionality:

- Processing raw data effectively
- Analysing and extracting data dynamically
- Providing recommendations and visualisation dynamically based on user requests

The weakness of the system is that the automatic data retrieval from HESA was not implemented but I have provided an alternative way to manually update the data in an easy way. Additionally, the project is not deployed, so it can only be used in the project environment.

8.2 Explaining LLM and Method Comparisons

Explanation of the Initial GPT-J Implementation and Reasons for its Failure

The system originally was designed to use GPT-J model which was supposed to be hosted locally and used for query interpretation. The implementation of that involved setting up the GPT-J model using Hugging Face Transformers library [9]. The model would send user queries to the local machine for data processing. However, this method encountered issue during implementation and testing which led to failure as a viable solution for this system.

The main reasons for its failure are:

- Performance Issues
    - Running GPT-J locally results in slow response time for query interpretation even on capable hardware. Unfortunately, it failed to meet performance goals.

- High Error Rates
    - For most of the results the data extracting from query interpretations were wrong, such as institution name and academic years.

- Output Inconsistencies
    - The output results were inconsistent, even running same query could results in different data extraction.

- Infrastructure Requirements
    - Hosting GPT-J demanded significant computational resources and technical effort for setup

Comparison of Approaches (Regex, GPT-J, Gemini)

- Accuracy
    - Regex: The accuracy was relatively good, but only for basic queries and only for certain type of data. When I was testing complex queries or when more data was

uploaded, its performance deteriorated significantly. A benefit of this approach was its speed, as it did not require any API calls

    - GPT-J: It performed better with more diverse data than the Regex approach, but it returned incorrect results most of the time. Also, this approach was the slowest of all tested because of hosting.

    - Gemini 1.5 Pro: Provides the most accurate and relatively quick results. Due to its larger capacity compared to the GPT-J and greater intelligence than the Regex approach. Its advanced natural language processing capabilities allow it to understand different phrasings and accurately extract the right data.

- Cost-Effectiveness
    - Regex: This approach is free to use because it is just combinations of different algorithms and patterns to find right data.

    - GPT-J: This model is free to use but it weighs about 50 GB, and it takes time when starting the server so this completely free, but it sacrifices the space and speed. Also, the speed of execution depends on the hardware that executes it, so the better hardware leads to performs better.

    - Gemini 1.5 Pro: The cost of Gemini Pro is different, it depends on the tokens used for API calls but Google provide a free credits that can be used to use the model. Refer to chapter 4.1 the overall cost of free-credits is around £3.88 so considering that Google provides £237 worth of free-credits it is cheap to utilize this model.

- Flexibility and Scalability
    - Regex: It provides no flexibility because it relies on predefined patterns and is therefore not scalable

    - GPT-J: It is flexible from the perspective of customizable configuration. It can be configured as the user wants, but its scalability is limited because of low capacity and outdated version.

    - Gemini 1.5 Pro: Even though Gemini is less customizable than GPT-J, its basic settings are sufficient to configure it for complex project. Furthermore, it is scalable because Google provide updates that yield even better results without requiring a complete change to the project structure.

On average, Gemini provided the best results comparing to the Regex and GPT-J approach. It is

not perfect, but it is well-suited for this type of project and complexity. It is easy to set up, Google provides free credits so it can be used for free of charge for significant amount of time and it is intelligent enough to perform complex data manipulation like HESA data.

8.3 Project Process Evolution and Timeline

The projected started with a structured plan as explained in Section 3.3 and early Gantt chart but the development process required evolution and adaptation. This section explains how the project timeline and approach changed from the initial plan to where it is right now. (see Figures 54,55)

The main reason for the project evolution was the decision to shift from the initially planned GPT-J model to Gemini 1.5 Pro API. The reason for initial implementation with GPT-J were its accessibility and no cost through self-hosting but it faced limitations in the early stage of devilment. These issues are further detailed in the section 8.2 includes:

- Including slow performance
- High error rate
- Limited understanding of HESA data
- Inconsistent output formats

This drastic change initially delayed the core query processing and dataset matching functionality because the main underlying AI technology was swapped. However, the superior capabilities of Gemini enabled better accuracy and opened possibilities for features that were not feasible for GPT-J. This led to a better overall outcome than initially expected, despite the re-work required

Another important change impacting the project timeline was the work on automated data updates. The initial goal was to allow the user to be able to automatically download the new data from HESA. Due to HESA's limited functionality for external usage, this was replaced with manual data updates

The iterative nature of the development process in integrating and refining the LLM interaction means that milestones were adjusted based on testing outcomes and the need to address unexpected issues.

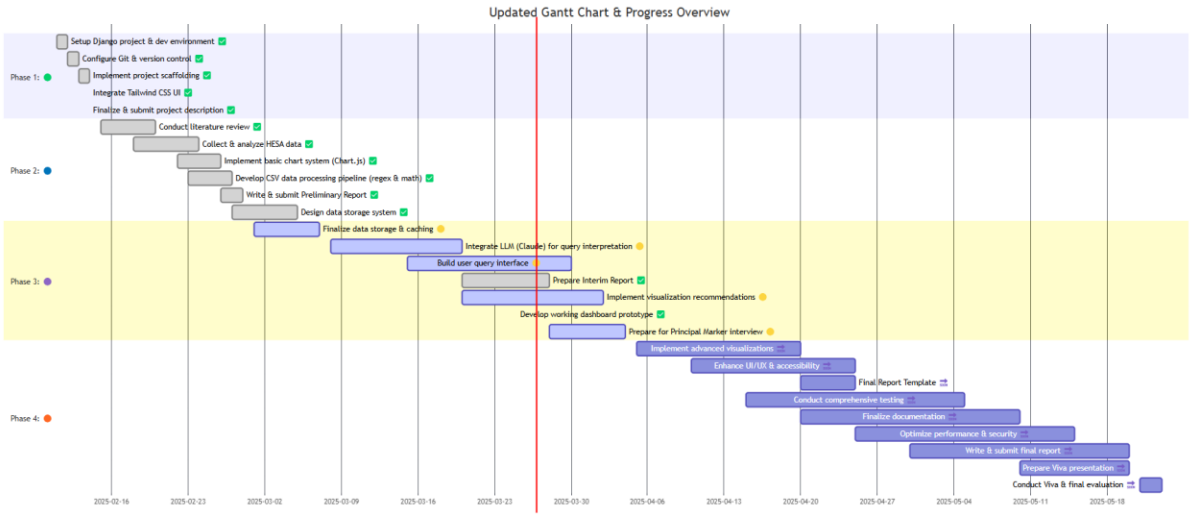


Figure 54: Revised Project Timeline - Gantt Chart

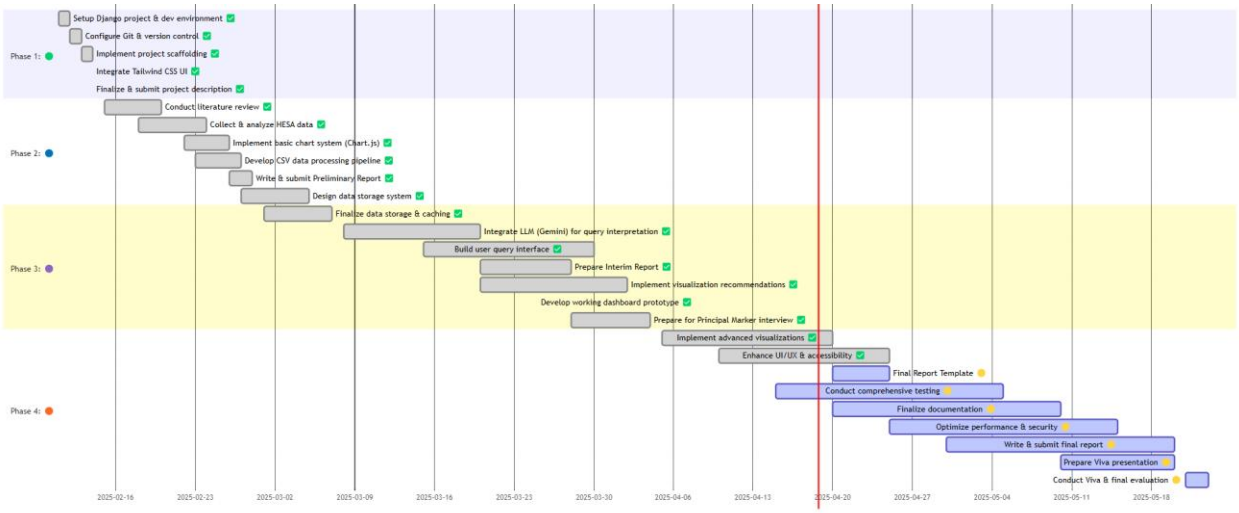


Figure 55: Final Project Timeline - Gantt Chart

Impact of Process Changes

After switching, I had to remove components primarily related to GPT-J, clean up the project and prepare it for the new LLM approach. Additionally, I had to re-implement the data extraction approach and processing techniques, as well as *csv_cleaning.py* functionality. I improved the cleaning files technique, so it creates indexing file which improves data processing and extraction. Consequently, this entire process took me more time, causing my original plan to shift, as I had to spend time re-doing tasks. The outcome was better than I expected because it not only fixed the issue faced when working with GPT-J but also it opened the possibility of implementing other features that were not feasible initially.

Lessons Learned from Process Evolution

I learnt from this process evolution the need to carefully analyse the core technology before starting using them and the importance of flexibility in the project planning. The importance of conducting more detailed research on the data that is required for the project before choosing what data fits best for the project. Learn more about technology's capabilities and limitations. Do more search about requirements chosen from the project to make sure they are implementable. In my case, the automatic data retrieval was not feasible to implement due to HESA limitations.

8.4 Achievement of Requirements

Most of the requirements that were outlined originally were implemented.

- Essential: 5/5
- Recommended: 9/10
- Optional: 1/7

Additionally, the essential requirement "Automated Data Retrieval" had to be cancelled as it was impossible to implement due to lack of HESA provided functionality. I decided to change it to "Manual data retrieval" as it still allows to perform data updates for the project in quick manner.

Priority	Requirement	Status
Essential	**Natural Language Query Interpretation:** The system understands users free-from queries and retrieve, and process data based on that.	✓
	User types a query in the search input, and after pressing the submit button, that query goes through different complex algorithms and returns data for the submitted query.	
	Integrating and Processing Data: Extracting and preparing clean HESA data, making it usable for comparing metric performance	✓
	The process data for the submitted query is being cleaned beforehand, therefore it is usable in comparison	
	Interactive Dashboard: It must dynamically display results in different format like tables, charts, diagrams or summary reports.	✓
	Displays data in Pie chart, Line chart or Bar char format	
	Comparative Analysis: A feature that must allow to group and benchmark universities by different categories like Mission Groups	✓
	The UI (User Interface) provides filtering options for Mission Group. There are 4 options: None (if no filtering should be applied), Russell Group, Million+ and University Alliance	
	Data Export Capabilities: Allow users to download generated reports and data as CSV, PFD or Excel file	

	After the user submits a query and selects a dataset, the user can download data for that selected dataset in 3 formats: CSV, PDF or Excel just be pressing a button	✓
Recommended:	**Automated Data Retrieval:** Make a way to automatically update the dashboard with new data from HESA when it is available.	X
	After some time researching, I realised that HESA does not provide any API or any other way to automatically download the data. The worst part is that they do not even say the exact date of when new data will be available, so I could create some sort of script to call the endpoint when new data becomes available. HESA says that they provide the date when new data will be available 4 weeks before the release of data. So, combining all the encountered issues, I decided to cancel this requirement and instead implemented manual data update, which was successfully implemented.	
	Using AI To Improve Analysis: LLM should provide insights, trend identification, and recommendations if user asks for that.	✓
	On the selected dataset page, after the user chooses to generate a visualisation at the bottom of that page, it also automatically generates the "Insights" section by Gemini based on the selected dataset and visualisation requirements. In that section, Gemini provide summarisation and key points of the generated visualisation. Also, on the selected dataset page, the Gemini analyses the dataset and provides recommendations for which type of visualisation is better for the selected dataset and requirements. If the user decides to generate a new visualization of a different type, Gemini will re-generate recommendations, visualization and insights, but it will notify the user that the newly selected chart type is not recommended for the selected dataset and explain the reason. In case a new, not recommended by Gemini visualisation cannot be generated, it will provide a specific message.	
	Query Building Guide: Provide a guide or pre-build example for users that are unfamiliar with the free-text queries	✓
	On the main page, there is a "Sample Questions" button that provides 10 queries of different formats and types that the user can try out. Also, it provides guides on how the patterns for searching data work and an explanation of the Mission Group	

	Advanced Visualisation: Support interactive graphs, trends projections that offer visual comparison of performance metrics.	✓
	The generated visualisations are not static but rather dynamic. The user can toggle data on generated charts by removing or putting back the data. Also, in the requirements, input user can ask Gemini to swap axis for a generated chart or to change colours	
	Historical Data Tracking: Tracking data changes overtime to see how organisational performance was changing relative to peers (for data processing)	✓
	The system handles multi-year data during the filtering and aggregation step, as well as generates a line chart that provides trends over time.	
	Caching for Frequent Queries: Implement caching to perform repeated queries more efficient that will result in better performance	✓
	Submitted query is now cached and stored in the cached folder in the format of a JSON document. It contains all the information about the submitted query. The cached data is saved for the period of 30 days, after that time, the cached data is removed.	
	LLM Interpretation Feedback: Allow users to see how their query was interpreted by the system to improved data transparency and remove bias factor	✓
	The submitted query provides information on why a particular dataset matches the query. It shows academic year/s were used to search for data, matching score and even names of the files from which where data.	
	Queries Logs: Log out user queries to improve future prompts tuning	✓
	User queries are getting logged out and stored along with the feedback that the user provides. It can be further used to analyse the popularity of searched data, the phrasing that users use and more.	
	Data Overload protection: Implement a safeguard that prevent outputs of big queries to potentially crush the system	✓
	Returned data is being truncated to only show the user a maximum of 3 rows of data. This preview mode goes for all the returned datasets which resulting in quicker and more efficient output.	

	CSV File Validation: Make a mechanism to automatically sanitize new raw HESA data (checking for correct format and size anomalies) to prepare it for the data processing.	✓
	The data originally provided in the raw format is then cleaned to be ready to be used for data processing. After cleaning data being stored in the "/cleaned_files" folder	
Optional:	**Predictive Modelling:** Using historical data and machine learning (ML) to predict future universities performance metrics	X
	The requirement was not implemented due to initial planning oversights and time constraints. The research identified relevant libraries, but insufficient time remained for their comprehensive understanding and integration.	
	Chatbot Integration: Include an AI chatbot assistant to help user to refine their queries dynamically, in real time.	X
	I did not create an AI chatbot that users could use for assistance, instead users can use fields for submitted visualisation specification, continuing therefore receive continuous feedback and greater results.	
	API for External Use: Create an API so that external and third-party application can use dashboard's functionality	X
	The project ended up being more complicated than I originally thought. Especially, regarding the creation of an API for external use. In fact, the creation of an API is primarily optional requirement because it does not affect the main purpose and idea of the project	
	Collect feedback: Implement mechanisms that would collect users' feedback through the application surveys to continuously	✓
	After the user submits a query and gets results, there is an option to provide feedback about the returned datasets. There are 3 options "Not Helpful", "Helpful" and "Very Helpful". Choosing any options but "Very Helpful" would require the user to provide a feedback comment (optional)	
	Data Storage Scalability: If HESA data volume grows beyond local CSV storage it will be migrated to a more scalable database solution such as (PostgreSQL or SQLite)	X
	The reason I included this requirement is that I was not sure whether I would need to relocate HESA data into something more scalable. Since I decided that storing data locally is good enough, implementing this requirement is pointless.	

	Multi-User Concurrency: Allow users to submit multiple queries simultaneously by using asynchronous job queues (such as Celery) and session management.	X
	It requires deployment, and since deployment is not implemented, this requirement cannot be achieved. This is not a critical requirement because it can be implemented later on, and this project can be used without multiple users using it at the same time	
	Deployment: Develop a deployment plan starting with local deployment and extending to containerization (Docker) and orchestration (Kubernetes) for future scalability.	X
	After research, I realised that the suggested way to deploy a project can only be done using the paid method, and I am not willing to spend my own money on a university project. Also, the deployment is not a critical requirement because it can be done at any point in time.	

Table 9: Project Requirements Implementation Status

8.5 Personal Learning and Reflection

These 3 months of project duration gave me significant personal and technical growth. Beyond specific technical skills that I gained, the process gave me valuable lessons in project management and problem solving.

A key area of learning was working with LLM for the first time in practical application. This includes the challenges faced with GPT-J and eventual integration of Gemini API. I learn more about both LLM model, specifically about what they offer and their limitations.

Working with HESA datasets was valuable experience in handling real world data challenges. I gain more knowledge about structure of HESA and complexities of cleaning, standardizing and preparing that data for analyses. The idea of creating the indexing file is clear example of understanding the need for efficient data handling.

Furthermore, this project was my only second experience using Django development stack. Working with this project improved familiarity with its key components and improved skills with backend. First ever experience with Chart.js was successful and I quickly become proficient in generating dynamic and interactive data visualization.

Beyond the core project technologies, I leveraged generative AI tools to help in the development process. These tools significantly contributed to my learning and efficiency:

- Grammarly AI: Used extensively to automatically check text for grammatical and syntactical mistakes. It helped me to refine the clarity and professionalism of project documentation and reports.

- GitHub Copilot: Integrated into VS Code which I used for developing this project. This tool provided:
    - Automated code suggestions (particularly helpful for boilerplate code)
    - Syntax recall
    - Exploring different implementation approaches
    - Automatic comment suggestion

- ChatGPT: Used for brainstorming ideas about the design of the website, search engine to quickly find information and to explain complicated concepts in simple way. Also, used other GPTs that help me in image and icon generation.

- Gemini 1.5 Pro: Served as a core functional component of the system for data processing. I have used Gemini to ask it to generate prompts that would get sent to Gemini API as detailed explanations of how to retrieve data or process it because I noticed that the prompts that were generated by Gemini worked better than the ones that I manually wrote myself. So, I used Gemini to get prompts and then manually tailor it if needed. (see Appendix F.3)

8.6 Critical Evaluation

Research

The literature review provided a better understanding of the technologies used in this project, such as the Chart.js documentation and the startup code for the LLM. I did not have any problems in finding the right data for my project, the only exception would be topics that are not publicly available, like the exact method that the University of Leicester is using when working with HESA data. It took me some time to understand the way HESA data is structured and how I can efficiently use it for data processing.

Design

The project architecture that I have chosen originally was perfect from the start, so I did not change anything about it as the project evolved. I did add more design features, a hide button for the Gemini analysis logic that is used for transparency, so the user could see how Gemini works. The button just allows the user to toggle the visibility of this information, resulting in a less cluttered webpage interface.

Implementation

I am proud of the CSV cleaning mechanism that I have implemented because it is multifunctional. It cleans the raw data and locates it to the right folder, then it extracts metadata that is used for simplifying the data processing functionality. Lastly, it creates the indexing file that is used to minimise the number of API calls required to find the right data.
The main challenge was to correctly implement the functionality so that it finds the right data based on the user's request. I tried many approaches to use the GPT-J, which failed, the Regex approach, which could help in case there is an issue with the Gemini API, but it was not dynamic enough, so I had to remove it, and only the Gemini 1.5 Pro managed to provide consistently correct results. The most helpful way to find the right solution was to debug my code, so I always make sure to logout unexpected results to observe the system behaviour during the execution.

8.7 Discussion of the Number of Requests and Associated Costs

During the implementation and testing phases, 882 requests were made to the Gemini 1.5 API. The volume of API calls made to Gemini 1.5 Pro over the project timeline. This includes APIs call to process, find, explain (insights) and visualize data (see Figure 56)

Figure 56: Volume of API calls made to Gemini 1.5 Pro over the project timeline.

The total number of API calls and associated cost metrics during the project. The first API call was made on March 17, and testing continued until April 29. The total number of requests made during this period was 882. The associated cost for these requests was approximately £3.88, which was covered by Google's free credit tier for API usage. (see Figure 57)

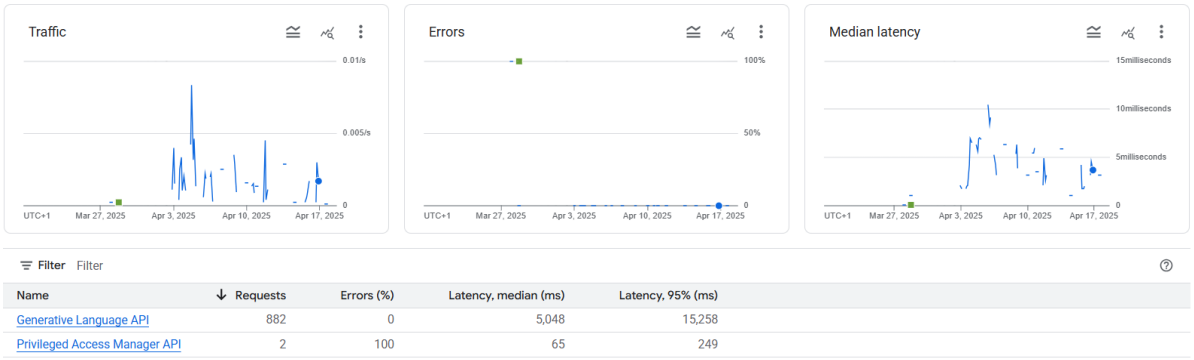


Figure 57: Associated cost metrics about the project

9. Conclusion and Future Work

9.1 Summary of Project Achievements

The project delivered the LLM-powered, dynamic visualization dashboard for the University of Leicester to easily work with HESA data. Currently, the project contains 392 CSV HESA cleaned files, which are all the relevant files up to date. It uses integrated Google Gemini 1.5 Pro to interpret free-text queries into structured parameters. Using Chart.js configuration to generate charts on demand. Created a responsive and nicely looking User Interface with feedback collection and data upload feature. The system is fully tested, and it has a 100% pass rate.

9.2 Conclusions

The system successfully overcomes the limitations of the manual and static original approach by:

- Reducing technical barriers and allowing a user without preparation to ask complex benchmarking questions in plain English
- Improving flexibility by enabling dynamic filtering, chart-type generation and changing without rebuilding pipelines.
- Enhancing the efficiency by cutting the time from a question to insights, from hours to seconds.

9.3 Future Work and Recommendations

To extend and strengthen the dashboard, some things can be done:

Broader HESA coverage & automation

- Automate data ingestion directly if HESA ever provide a public API and update the system so it can work even with detailed CSV files.

Predictive analytics & modelling

- Integrate Machine Learning (ML) driven trend projection so the user can see future trend changes, therefore gain more insights on the data.

Creating Administrator Panel

- The panel would allow the administrator to access the stored data directly through the User Interface (UI) and remove files or even data in case something was uploaded incorrectly

10. References

[1]. EleutherAI, 'GPT-J 6B', Hugging Face, 2021. [Online]. Available: https://huggingface.co/EleutherAI/gpt-j-6B (Accessed: 22 February 2025).

[2]. Brown, T. B. et al., 'Language Models are Few-Shot Learners', in Advances in Neural Information Processing Systems, 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html (Accessed: 22 February 2025).

[3]. Vaswani, A. et al., 'Attention is All You Need', in Advances in Neural Information Processing Systems, 2017, pp. 5998–6008. [Online]. Available: https://papers.nips.nips.cc/paper/7181-attention-is-all-you-need.pdf (Accessed: 22 February 2025).

[4]. Mitchell, R., Web Scraping with Python: Collecting Data from the Modern Web, 2nd ed., O'Reilly Media, 2018.

[5]. Higher Education Statistics Agency, 'HESA Data and Analysis', Higher Education Statistics Agency, 2023. [Online]. Available: https://www.hesa.ac.uk/data-and-analysis (Accessed: 22

February 2025).
[6]. Django Software Foundation, Django Documentation, 2023. [Online]. Available: https://docs.djangoproject.com/en/4.2/ (Accessed: 22 February 2025).
[7]. McKinney, W., Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython, 2nd ed., O'Reilly Media, 2017.
[8]. Tailwind Labs, Tailwind CSS Documentation, 2023. [Online]. Available: https://tailwindcss.com/docs (Accessed: 22 February 2025).
[9]. Wolf, T. et al., 'Transformers: State-of-the-Art Natural Language Processing', in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 38–45, 2020.
[10]. Prometheus Authors, Prometheus Documentation, 2023. [Online]. Available: https://prometheus.io/docs/introduction/overview/ (Accessed: 23 February 2025).
[11]. Grafana Labs, Grafana Documentation, 2023. [Online]. Available: https://grafana.com/docs/grafana/latest/ (Accessed: 23 February 2025).
[12]. Sentry, Sentry for Python, 2023. [Online]. Available: https://docs.sentry.io/platforms/python/ (Accessed: 24 February 2025).
[13] https://le.ac.uk/policies/privacy/students/student-information
[14] https://jobs.le.ac.uk/popups/displayfile.aspx?StoredFilePathID=wjMZlU3HiW3WxTamN-cedQ
[15] Kuc-Czarnecka, M., et al. (2024) Ethical Considerations in Data Collection and Analysis: A Review: Investigating Ethical Practices and Challenges in Modern Data Collection and Analysis. Available at: https://www.researchgate.net/publication/378789304_ETHICAL_CONSIDERATIONS_IN_DATA_COLLECTION_AND_ANALYSIS_A_REVIEW_INVESTIGATING_ETHICAL_PRACTICES_AND_CHALLENGES_IN_MODERN_DATA_COLLECTION_AND_ANALYSIS (Accessed: 22 April 2025).
[16] Sobha, D. N. K., et al. (2017) The Use of Quantitative Research Method and Statistical Data Analysis in Dissertation: An Evaluation Study. Available at: https://www.researchgate.net/publication/319468820_THE_USE_OF_QUANTITATIVE_RESEARCH_METHOD_AND_STATISTICAL_DATA_ANALYSIS_IN_DISSERTATION_AN_EVALUATION_STUDY (Accessed: 22 April 2025).
[17] Commission on Higher Education Policy Advice (IHEC) (2024) Data Matters in Higher Education. Available at: https://ihecommission.uk/wp-content/uploads/2024/06/IHEC_Data-Matters_03.06.2024.pdf (Accessed: 22 April 2025).
[18] Department of Education, Australia (2023) Benchmarking Cost Efficiency and Productivity in Universities. Available at: https://www.education.gov.au/download/18234/accord-report-benchmarking-cost-efficiency-and-productivity-universities/37655/document/pdf (Accessed: 22 April 2025).
[19] OWASP (2025) OWASP Top 10 for LLM Applications 2025. Available at: https://wtit.com/blog/2025/04/17/owasp-top-10-for-llm-applications-2025/ (Accessed: 22 April 2025).

[20] Nagaraja, N. and Bahsi, H. (2025) Cyber Threat Modeling of an LLM-Based Healthcare System. Available at: https://www.scitepress.org/Papers/2025/132897/132897.pdf (Accessed: 22 April 2025).
[21] Sun, Y., et al. (2024) Natural Language Interfaces for Tabular Data Querying and Visualization: A Survey. Available at: https://www.researchgate.net/publication/380581799_Natural_Language_Interfaces_for_Tabular_Data_Querying_and_Visualization_A_Survey (Accessed: 22 April 2025).
[22] Sankhla, H. (2025) Next-Generation Database Interfaces: A Comprehensive Survey of LLM-Based Text-to-SQL. Available at: https://dev.to/hardiksankhla/next-generation-database-interfaces-a-comprehensive-survey-of-llm-based-text-to-sql-20i (Accessed: 22 April 2025).
[23] Google, 'Our next-generation model: Gemini 1.5', Google Blog, 2024. [Online]. Available: https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/ (Accessed: 2 April 2025).
[24]. Higher Education Statistics Agency, 'HESA Data and Analysis', Higher Education Statistics Agency, 2023. [Online]. Available: https://www.hesa.ac.uk/data-and-analysis/upcoming (Accessed: 2 May 2025).

11. Appendix

A: Pandas-Based Data Processing Pipeline

A.1 Data Transformation for Visualization

```python
def transform_chart_data(csv_path):
    """
    Transform the CSV data into the required format for charts.
    Expected columns after transformation: Academic Year, Level of study, Number
    """
    # Read the CSV file
    df = pd.read_csv(csv_path)

    # Skip metadata rows (first 6 rows)
    data_start = 6
    df = pd.read_csv(csv_path, skiprows=range(data_start))

    # Rename columns if needed
    if 'Year' in df.columns:
        df = df.rename(columns={'Year': 'Academic Year'})

    # Ensure required columns exist
    required_columns = ['Academic Year', 'Level of study', 'Number']

    # If the data is in wide format (years as columns), melt it to long format
    if not all(col in df.columns for col in required_columns):
        # Identify the year columns (assuming they follow a pattern like YYYY/YY)
        year_columns = [col for col in df.columns if '/' in str(col)]

        if year_columns:
```

```
            # Melt the dataframe to convert years to rows
            df = pd.melt(
                df,
                id_vars=[col for col in df.columns if col not in year_columns],
                value_vars=year_columns,
                var_name='Academic Year',
                value_name='Number'
            )

    # Clean up the data
    df = df.dropna()
    df['Number'] = pd.to_numeric(df['Number'], errors='coerce')

    # Verify required columns exist
    missing_columns = [col for col in required_columns if col not in df.columns]
    if missing_columns:
        raise ValueError(f"Missing required columns: {', '.join(missing_columns)}")

    return df
```

A.2 Multi-format Data Loading in Correlation Engine

```
def load_file(self, file_path: str) -> pd.DataFrame:        You, 2 months ago •
    """
    Load a data file into a DataFrame based on its extension.
    """
    file_path = Path(file_path)

    if not file_path.exists():
        raise FileNotFoundError(f"File not found: {file_path}")

    # Load based on file extension
    if file_path.suffix.lower() == '.csv':
        return pd.read_csv(file_path)
    elif file_path.suffix.lower() in ['.xlsx', '.xls']:
        return pd.read_excel(file_path)
    elif file_path.suffix.lower() == '.json':
        return pd.read_json(file_path)
    elif file_path.suffix.lower() == '.pkl':
        return pd.read_pickle(file_path)
    else:
        raise ValueError(f"Unsupported file format: {file_path.suffix}")
```

A.3 Chart Data Preparation with Pandas Pivot

```
49  def prepare_chart_data(df, chart_type):
50      """
51      Prepare the data for different chart types
52      """
53      if chart_type == 'line':
54          # Group by Academic Year and Level of study
55          data = df.pivot(index='Academic Year', columns='Level of study', values='Number')
56          return {
57              'labels': data.index.tolist(),
58              'datasets': [
59                  {
60                      'label': level,
61                      'data': data[level].tolist(),
62                      'fill': False,
63                      'borderColor': f'hsl({(i * 360/len(data.columns))}, 70%, 50%)'
64                  }
65                  for i, level in enumerate(data.columns)
66              ]
67          }
68
69      elif chart_type == 'bar':
70          # Group by Level of study and sum the numbers
71          data = df.groupby('Level of study')['Number'].sum().sort_values(ascending=False)
72          return {
73              'labels': data.index.tolist(),
74              'datasets': [{
75                  'label': 'Total Students',
76                  'data': data.values.tolist(),
```

```
                'backgroundColor': [
                    f'hsl({(i * 360/len(data))}, 70%, 50%)'
                    for i in range(len(data))
                ]
            }]
        }

    elif chart_type == 'pie':
        # Group by Level of study and sum the numbers
        data = df.groupby('Level of study')['Number'].sum()
        return {
            'labels': data.index.tolist(),
            'datasets': [{
                'data': data.values.tolist(),
                'backgroundColor': [
                    f'hsl({(i * 360/len(data))}, 70%, 50%)'
                    for i in range(len(data))
                ]
            }]
        }

    else:
        raise ValueError(f"Unsupported chart type: {chart_type}")
```

B: Unit Testing

B.1, B.2 Data Processing and LLM Integration

```
class TestQueryParser:
    """Test cases for query parsing functionality."""

    Windsurf: Refactor | Explain | ×
    def test_parse_hesa_query_basic(self):
        """Test basic query parsing functionality."""
        # Test a simple query
        query = "How many students were at University of Testing in 2020?"

        # Mock the parse_hesa_query function
        with patch('core.views.process_gemini_query') as mock_process:
            # Mock the response from process_gemini_query
            mock_process.return_value = {
                'institutions': ['University of Testing'],
                'years': ['2020'],
                'data_request': ['student', 'enrollment']
            }

            # Mock the parse_hesa_query function
            mock_parse_query = MagicMock()
            mock_parse_query.return_value = {
                'institutions': ['University of Testing'],
                'years': ['2020'],
                'data_request': ['student', 'enrollment']
            }

            result = mock_parse_query(query)

```

```
            # Check the result
            assert result is not None, "Result should not be None"
            assert 'institutions' in result, "Result should include institutions"
            assert 'years' in result, "Result should include years"
            assert 'data_request' in result, "Result should include data_request"
            assert 'University of Testing' in result['institutions'], "Should extract University of Testing"
            assert '2020' in result['years'], "Should extract year 2020"
            assert 'student' in result['data_request'], "Should identify student data request"

    Windsurf: Refactor | Explain | ×
    def test_local_analyze_query(self):
        """Test the local query analysis fallback."""
        # Test a simple query
        query = "Show me undergraduate students at University of Leicester in 2019"
              You, 2 weeks ago • created unit testing for CSV Processor, Query P…
        # Use the mock AI client directly
        mock_client = MockAIClient()
        result = mock_client.analyze_query(query)

        # Check the result
        assert result is not None, "Result should not be None"
        assert 'institutions' in result, "Result should include institutions"
        assert 'years' in result, "Result should include years"
        assert 'data_request' in result, "Result should include data_request"

        # Check specific extractions
        assert any('Leicester' in inst for inst in result['institutions']), "Should extract University of Leicester"
        assert any('2019' in year for year in result['years']), "Should extract year 2019"
        assert any('undergraduate' in req.lower() for req in result['data_request']), "Should identify undergraduate request"
    def test_query_with_multiple_institutions(self):
        """Test parsing a query with multiple institutions."""
        # Test a query with multiple institutions
        query = "Compare student numbers at University of Testing and Example University for 2020"

        # Use the mock AI client directly
        mock_client = MockAIClient()
        result = mock_client.analyze_query(query)

        # Check the result
        assert len(result['institutions']) >= 2, "Should extract at least 2 institutions"

        # Check for the presence of institutions (case-insensitive)
        institutions_lower = [inst.lower() for inst in result['institutions']]
        assert any('testing' in inst for inst in institutions_lower), "Should extract University of Testing"
        assert any('example' in inst for inst in institutions_lower), "Should extract Example University"

    Windsurf: Refactor | Explain | ×
    def test_query_with_year_range(self):
        """Test parsing a query with a year range."""
        # Test a query with a year range
        query = "Show undergraduate enrollment at University of Leicester from 2018 to 2020"

        # Use the mock AI client directly
        mock_client = MockAIClient()
        result = mock_client.analyze_query(query)

        # Check the result
        assert result['start_year'] is not None, "Should identify a start year"
        assert result['end_year'] is not None, "Should identify an end year"
        assert result['start_year'] == '2018', "Start year should be 2018"
        assert result['end_year'] == '2020', "End year should be 2020"

    Windsurf: Refactor | Explain | ×
    def test_query_with_academic_year(self):
        """Test parsing a query with academic year format."""
        # Test a query with academic year format
        query = "How many students were enrolled in 2019/20 at University of Leicester?"

        # Use the mock AI client directly
        mock_client = MockAIClient()
        result = mock_client.analyze_query(query)

        # Check the result - the mock might not handle academic years perfectly but should extract 2019
        assert any('2019' in year for year in result['years']), "Should extract year 2019"
```

B.3 Visualization

```python
class TestChartGeneration:      You, 2 weeks ago • created unit testing for CSV Processor, Query P…
    """Test cases for chart generation functionality."""

    Windsurf: Refactor | Explain | ×
    @pytest.fixture
    def mock_client(self):
        """Create a mock Gemini client."""
        mock = MagicMock()

        # Mock generate_content method
        mock.models.generate_content.return_value = MagicMock()
        mock.models.generate_content.return_value.text = json.dumps({
            "recommended_chart_type": "bar",
            "recommendation_reason": "A bar chart is recommended for comparing values across categories.",
            "example_prompts": [
                "Compare undergraduate students across universities",
                "Show postgraduate numbers by institution",
                "Display student counts by degree type"
            ]
        })

        return mock

    Windsurf: Refactor | Explain | ×
    def test_get_chart_recommendation(self, mock_client, sample_visualization_data):
        """Test getting chart recommendations."""
        # Set up the mock to return a specific result
        mock_get_chart_recommendation.return_value = {"success": True}
        # Call the function with mock client and test data
        with patch('core.views.CustomJsonResponse') as mock_response:
            mock_response.return_value = {"success": True}

            # Call our mock function instead of the real one
            response = mock_get_chart_recommendation(mock_client, sample_visualization_data)

            # Check response
            assert response is not None
            assert response == {"success": True}

    Windsurf: Refactor | Explain | ×
    def test_generate_visualization(self, mock_client, sample_visualization_data):
        """Test generating a visualization."""
        # Mock client response for visualization generation
        mock_client.models.generate_content.return_value.text = """
        ```json
        {
            "chart_config": {
                "type": "bar",
                "data": {
                    "labels": ["University of Testing", "Example University", "Sample College"],
                    "datasets": [{
                        "label": "Undergraduate Students",
                        "data": [6000, 5300, 3600],
                        "backgroundColor": ["#4e73df", "#1cc88a", "#36b9cc"]
                    }]
                },
```
```

```
            "options": {
                "responsive": true,
                "plugins": {
                    "title": {
                        "display": true,
                        "text": "Undergraduate Students by University"
                    }
                }
            }
        },
        "insights": "<p>The chart shows that University of Testing has the highest number of undergraduate students.</p>"
    }
    ```
    """

    # Set up the mock to return a specific result
    mock_extract_json = MagicMock()
    mock_extract_json.return_value = {
        "chart_config": {
            "type": "bar",
            "data": {
                "labels": ["University of Testing", "Example University", "Sample College"],
                "datasets": [{
                    "label": "Undergraduate Students",
                    "data": [6000, 5300, 3600],
                    "backgroundColor": ["#4e73df", "#1cc88a", "#36b9cc"]
                }]
            },
            "options": {
                "responsive": True,
                "plugins": {   You, 2 weeks ago • created unit testing for CSV Processor, Query P…
                    "title": {
                        "display": True,
                        "text": "Undergraduate Students by University"
                    }
                }
            }
        },
        "insights": "<p>The chart shows that University of Testing has the highest number of undergraduate students.</p>"
    }

    mock_generate_visualization.return_value = {"success": True}

    # Test generating a visualization
    with patch('core.views.extract_and_sanitize_json') as mock_extract:
        # Mock the extracted JSON
        mock_extract.return_value = mock_extract_json.return_value

        with patch('core.views.CustomJsonResponse') as mock_response:
            mock_response.return_value = {"success": True}

            # Call the function with mock client and sample data
            request = "Show undergraduate students by university"
            response = mock_generate_visualization(mock_client, sample_visualization_data, request)

            # Check response
            assert response is not None
            assert response == {"success": True}
```
```

```python

    Windsurf: Refactor | Explain | ×
    def test_change_chart_type(self, mock_client, sample_visualization_data):
        """Test changing a chart type."""
        # Mock client response for chart type change
        mock_client.models.generate_content.return_value.text = """
        ```json
        {
            "recommended_chart_type": "line",
            "recommendation_reason": "A line chart is recommended for showing trends over time.",
            "example_prompts": [
                "Show trends in student numbers over time",
                "Display changes in enrollment across years",
                "Visualize growth in postgraduate students"
            ]
        }
        ```
        """

        # Create an original recommendation
        original_recommendation = json.dumps({
            "recommended_chart_type": "bar",
            "recommendation_reason": "A bar chart is recommended for comparing values across categories.",
            "example_prompts": [
                "Compare undergraduate students across universities",
                "Show postgraduate numbers by institution",
                "Display student counts by degree type"
            ]
        })

        # Set up the mock to return a specific result
        mock_change_chart_type.return_value = {"success": True}

        # Test changing chart type
        with patch('core.views.CustomJsonResponse') as mock_response:
            mock_response.return_value = {"success": True}

            # Call the function
            response = mock_change_chart_type(
                mock_client,
                sample_visualization_data,
                "line",
                original_recommendation
)

            # Check response
            assert response is not None
            assert response == {"success": True}

    Windsurf: Refactor | Explain | ×
    def test_chart_type_explanation(self):
        """Test getting chart type explanations."""
        # Set up mock returns
        mock_get_chart_type_explanation.side_effect = lambda chart_type, dataset_characteristics: {
            "line": "Line charts are best for showing trends over time.",
            "pie": "Pie charts are best for showing parts of a whole with a small number of categories."
        }.get(chart_type, "")
```

```
        # Test with line chart and single-year dataset
        chart_type = "line"
        dataset_characteristics = {
            "has_multiple_years": False,
            "has_multiple_institutions": True,
            "number_of_numeric_columns": 5,
            "total_rows": 10,
            "available_institutions": ["University of Testing", "Example University"],
            "available_years": ["2020/21"]
        }

        explanation = mock_get_chart_type_explanation(chart_type, dataset_characteristics)

        # Check result
        assert explanation is not None
        assert "line" in explanation.lower()
        assert "trend" in explanation.lower()
        assert "time" in explanation.lower()

        # Test with pie chart and too many categories
        chart_type = "pie"
        dataset_characteristics["number_of_numeric_columns"] = 8

        explanation = mock_get_chart_type_explanation(chart_type, dataset_characteristics)

        # Check result
        assert explanation is not None
        assert "pie" in explanation.lower()
```

C: Integration tests

C.1 Query to Visualization Flow

```python
class TestQueryToVisualizationFlow:      You, 2 weeks ago • created intergration test for Mock AI Client In…
    """Test cases for the end-to-end flow from query to visualization."""

    Windsurf: Refactor | Explain | ×
    @pytest.fixture(scope="class")
    def csv_processor(self):
        """Fixture for the CSV processor."""
        return CSVProcessor()

    Windsurf: Refactor | Explain | ×
    @pytest.fixture(scope="class")
    def mock_client(self):
        """Fixture for the mock AI client."""
        return MockAIClient()

    Windsurf: Refactor | Explain | ×
    @pytest.fixture(scope="class")
    def sample_cleaned_files(self, csv_processor):
        """Return a list of available cleaned files for testing."""
        available_files = csv_processor.get_available_datasets()
        if not available_files:
            pytest.skip("No cleaned CSV files available for testing")
        return available_files

    Windsurf: Refactor | Explain | ×
    def test_query_parse_to_data_retrieval(self, mock_client, csv_processor, sample_cleaned_files):
        """Test the flow from query parsing to data retrieval."""
        # Skip if no data is available
        if not sample_cleaned_files:
            pytest.skip("No sample data available for testing")

        # Use a sample query about student enrollment
        query = "How many students were enrolled at University of Cambridge in 2020/21?"

        # Process the query using the mock client
        query_analysis = mock_client.analyze_query(query)

        # Verify the query was analyzed correctly
        assert query_analysis is not None, "Query analysis should not be None"
        assert "institutions" in query_analysis, "Query analysis should extract institutions"
        assert "years" in query_analysis, "Query analysis should extract years"

        # Get institutions and years from the query
        institutions = query_analysis.get("institutions", [])
        years = query_analysis.get("years", [])

        assert len(institutions) > 0, "At least one institution should be extracted"
        assert len(years) > 0, "At least one year should be extracted"

        # Now try to find matching datasets based on the query
        matching_files = []
        for file_name in sample_cleaned_files:
            for institution in institutions:
                if institution.lower() in file_name.lower():
                    for year in years:
                        if year.replace("/", "-") in file_name:
                            matching_files.append(file_name)
                            break
```

```python
        # If we don't find exact matches, just check for any files with similar keywords
        if not matching_files:
            for file_name in sample_cleaned_files:
                # Check if it contains keywords like "student", "enrollment", etc.
                if any(keyword in file_name.lower() for keyword in ["student", "enrollment", "enrollment"]):
                    matching_files.append(file_name)
                    if len(matching_files) >= 2:  # Limit to 2 files for testing
                        break

        # If we still don't have matches, use the first two files
        if not matching_files and len(sample_cleaned_files) > 0:
            matching_files = sample_cleaned_files[:min(2, len(sample_cleaned_files))]

        assert len(matching_files) > 0, "Should find at least one matching file"

        # Get the metadata from the first matching file
        file_path = csv_processor.clean_dir / matching_files[0]
        metadata = csv_processor.extract_metadata(file_path)

        # Check that we can extract metadata from the file
        assert metadata is not None, "Should be able to extract metadata"

    Windsurf: Refactor | Explain | ×
    def test_data_to_visualization_flow(self, mock_client, csv_processor, sample_cleaned_files):
        """Test the flow from data retrieval to visualization generation."""
        # Skip if no data is available
        if not sample_cleaned_files:
            pytest.skip("No sample data available for testing")
        test_file = sample_cleaned_files[0]      You, 2 weeks ago • created intergration test for Mock AI Client In…
        file_path = csv_processor.clean_dir / test_file

        # Extract metadata
        metadata = csv_processor.extract_metadata(file_path)

        # Load the data
        df = csv_processor.clean_csv(file_path)
        assert df is not None, "Should be able to load the CSV data"

        # Prepare sample visualization data (simplified)
        visualization_data = {
            'title': metadata.get('title', 'Sample Dataset'),
            'columns': df.columns.tolist(),
            'rows': df.head(5).values.tolist()
        }

        # Get chart recommendation
        chart_recommendation = mock_client.get_chart_recommendation(visualization_data)

        # Verify chart recommendation
        assert chart_recommendation is not None, "Should get a chart recommendation"
        assert "recommended_chart_type" in chart_recommendation, "Should include a recommended chart type"
        assert "recommendation_reason" in chart_recommendation, "Should include a reason for recommendation"

        # Generate a visualization
        chart_config = mock_client.generate_visualization_config(
            visualization_data,
            "Show student data",
```

```
            chart_recommendation["recommended_chart_type"]
        )

        # Verify visualization generation
        assert chart_config is not None, "Should generate a visualization configuration"
        assert "chart_config" in chart_config, "Should include chart configuration"
        assert "insights" in chart_config, "Should include insights"

        # Verify chart configuration
        assert "type" in chart_config["chart_config"], "Chart config should include type"
        assert "data" in chart_config["chart_config"], "Chart config should include data"
        assert "options" in chart_config["chart_config"], "Chart config should include options"

Windsurf: Refactor | Explain | ×
    def test_end_to_end_query_processing(self, mock_client, csv_processor, sample_cleaned_files):
        """Test the complete end-to-end flow from query to visualization."""
        # Skip if no data is available
        if not sample_cleaned_files:
            pytest.skip("No sample data available for testing")

        # Sample natural language query
        query = "Show me student enrollments by university in 2022/23"

        # Step 1: Extract query parameters
        query_analysis = mock_client.analyze_query(query)

        # Verify query analysis
        assert query_analysis is not None, "Query should be analyzed"

        # Step 2: Find matching datasets (simulated)
        matching_files = []
        for file_name in sample_cleaned_files:
            # Look for enrollment-related files
            if any(term in file_name.lower() for term in ["enrolment", "enrollment", "student"]):
                # Look for the specific year
                if "2022-23" in file_name:
                    matching_files.append(file_name)
                    break

        # If no exact matches, use any enrollment-related file
        if not matching_files:
            for file_name in sample_cleaned_files:
                if any(term in file_name.lower() for term in ["enrolment", "enrollment", "student"]):
                    matching_files.append(file_name)
                    break

        # If still no matches, use the first file
        if not matching_files and sample_cleaned_files:
            matching_files = [sample_cleaned_files[0]]

        assert len(matching_files) > 0, "Should find at least one matching file"

        # Step 3: Load and prepare data
        file_path = csv_processor.clean_dir / matching_files[0]
        df = csv_processor.clean_csv(file_path)
        assert df is not None, "Should be able to load the CSV data"

        # Step 4: Prepare visualization data
```

```
        visualization_data = {
            'title': matching_files[0].replace('.csv', ''),
            'columns': df.columns.tolist(),
            'rows': df.head(10).values.tolist()
        }

        # Step 5: Get chart recommendation
        chart_recommendation = mock_client.get_chart_recommendation(visualization_data)
        assert "recommended_chart_type" in chart_recommendation, "Should recommend a chart type"

        # Step 6: Generate visualization
        chart_config = mock_client.generate_visualization_config(
            visualization_data,
            query,
            chart_recommendation["recommended_chart_type"]
        )

        # Verify complete flow output
        assert chart_config is not None, "Should generate a visualization"
        assert "chart_config" in chart_config, "Should include chart configuration"
        assert "insights" in chart_config, "Should include data insights"

        # Check that the chart config includes all necessary components
        chart_type = chart_config["chart_config"]["type"]
        assert chart_type in ["bar", "line", "pie", "scatter", "doughnut"], f"Chart type should be valid, got {chart_type}"
        assert "data" in chart_config["chart_config"], "Chart config should include data"
        assert "labels" in chart_config["chart_config"]["data"], "Chart data should include labels"
        assert "datasets" in chart_config["chart_config"]["data"], "Chart data should include datasets"
```

C.2 Mock AI Client Integration

```python
class TestMockAIClientIntegration:      You, 2 weeks ago • created intergration test for Mock AI Client In…
    """Test cases for the MockAIClient in integration scenarios."""

    Windsurf: Refactor | Explain | ×
    @pytest.fixture
    def mock_client(self):
        """Mock AI client fixture."""
        return MockAIClient()

    Windsurf: Refactor | Explain | ×
    def test_chart_recommendation(self, mock_client):
        """Test that chart recommendations work as expected."""
        # Create visualization data
        visualization_data = {
            'title': 'Student Enrollments 2020-21',
            'columns': ['University', 'Undergraduate', 'Postgraduate', 'Total'],
            'rows': [
                ['University A', '1000', '500', '1500'],
                ['University B', '2000', '1000', '3000'],
                ['University C', '3000', '1500', '4500']
            ]
        }

        # Test the chart recommendation function
        chart_recommendation = mock_client.get_chart_recommendation(visualization_data)

        # Assert that the mock client returns the expected structure
        assert "recommended_chart_type" in chart_recommendation, "Should include a recommended chart type"
        assert "recommendation_reason" in chart_recommendation, "Should include a reason for the recommendation"
        assert "example_prompts" in chart_recommendation, "Should include example prompts"
    def test_visualization_generation(self, mock_client):
        """Test visualization generation functionality."""
        # Create visualization data
        visualization_data = {
            'title': 'Student Enrollments 2020-21',
            'columns': ['University', 'Undergraduate', 'Postgraduate', 'Total'],
            'rows': [
                ['University A', '1000', '500', '1500'],
                ['University B', '2000', '1000', '3000'],
                ['University C', '3000', '1500', '4500']
            ]
        }

        # Generate a visualization config
        chart_config = mock_client.generate_visualization_config(
            visualization_data,
            "Show student enrollments",
            "bar"
        )

        # Verify the chart configuration structure
        assert chart_config is not None, "Should generate a visualization configuration"
        assert "chart_config" in chart_config, "Should include chart configuration"
        assert "insights" in chart_config, "Should include insights"

        # Basic test of chart configuration structure
        assert "type" in chart_config["chart_config"], "Chart config should include type"
        assert "data" in chart_config["chart_config"], "Chart config should include data"
        assert "labels" in chart_config["chart_config"]["data"], "Chart data should include labels"
```

```python
    def test_chart_type_change(self, mock_client):
        """Test changing chart type functionality."""
        # Create visualization data
        visualization_data = {
            'title': 'Student Enrollments 2020-21',
            'columns': ['University', 'Undergraduate', 'Postgraduate', 'Total'],
            'rows': [
                ['University A', '1000', '500', '1500'],
                ['University B', '2000', '1000', '3000'],
                ['University C', '3000', '1500', '4500']
            ]
        }

        # Generate a visualization with a different chart type
        chart_config = mock_client.generate_visualization_config(
            visualization_data,
            "Show student enrollments",
            "pie"  # Use pie chart instead of default bar
        )

        # Verify the chart type was changed
        assert chart_config["chart_config"]["type"] == "pie", "Chart type should be pie"
        assert "data" in chart_config["chart_config"], "Chart config should include data"
```

D: Data Processing Pipeline Implementation

D.1 Data Ingestion Process

```python
    def process_single_file(self, file_name: str) -> bool:      You, 2 months ago • using regular expression predified query
        """
        Process a single file by name from the raw directory.
        """
        file_path = self.raw_dir / file_name

        try:
            logger.info(f"Processing single file: {file_name}")

            # Validate file
            if not self.validate_csv(file_path):
                return False

            # Create new empty metadata
            metadata = {}

            # Read the first 20 lines to extract title and academic year
            try:
                with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
                    lines = [line.strip() for line in f.readlines()[:20]]

                # Extract title from the first line only
                first_line = lines[0] if lines else ""

                # Log the raw line for debugging
                logger.info(f"Raw first line: [{first_line}]")

                # Check if the line contains the title pattern
                if 'Title:' in first_line:
                    # Get everything after 'Title:'
                    title_text = first_line.split('Title:')[1].strip()
                    logger.info(f"After split by 'Title:': [{title_text}]")

                    # Remove leading commas and quotes
                    title_text = title_text.lstrip(',').strip('"')
                    logger.info(f"After cleaning commas/quotes: [{title_text}]")

                    # Find the part after "Table XX - " if it exists
                    table_match = re.search(r'Table\s+\d+\s*-\s*(.*)', title_text)
                    if table_match:
                        # Extract just the part after the dash
                        extracted_title = table_match.group(1).strip()
                        # Remove any trailing quotes
                        extracted_title = extracted_title.rstrip('"')
                        metadata['title'] = extracted_title
                        logger.info(f"Extracted title: [{extracted_title}]")
                    else:
                        # If no table pattern, just use the whole text
                        cleaned_title = title_text.strip()
                        metadata['title'] = cleaned_title
                        logger.info(f"No table pattern found, using cleaned title: [{cleaned_title}]")
                else:
                    logger.warning(f"First line does not contain 'Title:': [{first_line}]")

                # Extract academic year
                for line in lines:
                    # Skip lines with "to" as they indicate a range of years (e.g., "Academic years 2020/21 to 2023/24")
                    if "to" in line:
```

```
                    continue

                # Look for patterns like ",Academic year,YYYY/YY" or "Filters:,Academic year,YYYY/YY"
                year_match = re.search(r'(?:,|\:)\s*Academic year\s*,\s*(20\d{2}/\d{2})', line)
                if year_match:
                    metadata['academic_year'] = year_match.group(1).strip()
                    logger.info(f"Found academic year format: {metadata['academic_year']}")
                    break
        except Exception as e:
            logger.warning(f"Error extracting title/academic year: {str(e)}")

        # If title not found, use "Unknown title"
        if 'title' not in metadata:
            logger.warning(f"Could not extract title from file content")
            metadata['title'] = "Unknown title"

        # If academic year not found, use "Unknown"
        if 'academic_year' not in metadata:
            logger.warning(f"Could not extract academic year from file content")

            # Try to extract from filename before defaulting to Unknown
            year_match = re.search(r'(\d{4})[\.\-&_](\d{2})', file_name)
            if year_match:
                metadata['academic_year'] = f"{year_match.group(1)}/{year_match.group(2)}"
                logger.info(f"Extracted academic year from filename: {metadata['academic_year']}")
            else:
                metadata['academic_year'] = "Unknown"

        # Clean data
        cleaned_df = self.clean_csv(file_path)
        if cleaned_df is None:
            return False

        # Extract title-based keywords
        title_keywords = self.extract_keywords_from_title(metadata['title'])
        metadata['keywords_title'] = title_keywords

        # Extract column-based keywords
        column_keywords = self.extract_keywords_from_columns(cleaned_df.columns)
        metadata['keywords_columns'] = column_keywords

        # Generate new filename based on title and academic year
        original_filename = file_name
        new_filename = self.generate_clean_filename(metadata['title'], metadata['academic_year'], original_filename)

        # Store original filename in metadata
        metadata['original_filename'] = original_filename
        # Store new filename in metadata
        metadata['new_filename'] = new_filename

        # Create metadata line
        metadata_line = f"#METADATA:{json.dumps(metadata)}\n"

        # Save cleaned file with metadata using new filename
        cleaned_path = self.clean_dir / new_filename

        # Write metadata line first, then the data
        with open(cleaned_path, 'w', encoding='utf-8') as f:
            f.write(metadata_line)

        # Then append the dataframe
        cleaned_df.to_csv(cleaned_path, index=False, mode='a')

        logger.info(f"Successfully processed {file_name} with metadata")
        logger.info(f"Title: {metadata.get('title', 'Not found')}")
        logger.info(f"Academic Year: {metadata.get('academic_year', 'Not found')}")
        logger.info(f"New filename: {new_filename}")
        logger.info(f"Title keywords: {title_keywords}")
        logger.info(f"Column keywords: {column_keywords}")
        return True

    except Exception as e:
        logger.error(f"Error processing {file_name}: {str(e)}")
        return False

```

D.2 Manual Update Process

```python
@csrf_exempt
def upload_data_view(request):      You, 2 weeks ago • updated the loading indicator and created manua…
    """Render the upload data page."""
    return render(request, 'upload_data.html', {'show_form': True})

@csrf_exempt
@require_http_methods(["POST"])
def process_upload(request):
    """
    # Define BASE_DIR
    BASE_DIR = Path(__file__).resolve().parent.parent

    # Set up logging
    logger = logging.getLogger(__name__)
    logger.info("Processing file upload request")

    # Prepare response data
    response_data = {
        'success': False,
        'error': None,
        'total_files': 0,
        'added_files_count': 0,
        'added_files': [],
        'error_files': [],
        'cleaning_status': 'Not started',
        'indexing_created': False
    }

    try:
        # Get file processing option
        processing_option = request.POST.get('processing_option', 'add')
        logger.info(f"Processing option selected: {processing_option}")

        # Check if files were uploaded
        if 'csv_files' not in request.FILES:
            response_data['error'] = 'No files were uploaded'
            return JsonResponse(response_data)

        # Get uploaded files
        files = request.FILES.getlist('csv_files')
        logger.info(f"Received {len(files)} files for upload")

        # Check if uploaded files exceed the maximum allowed number
        max_files = getattr(settings, 'DATA_UPLOAD_MAX_NUMBER_FILES', 20)  # Default is 20
        if len(files) > max_files:
            logger.error(f"Too many files: received {len(files)}, max allowed is {max_files}")
            response_data['error'] = f"The number of files exceeded the maximum limit of {max_files}. Please upload
            return JsonResponse(response_data)

        # Validate files are CSVs
        non_csv_files = [f.name for f in files if not f.name.lower().endswith('.csv')]
        if non_csv_files:
            response_data['error'] = f"Only CSV files are allowed. Invalid files: {', '.join(non_csv_files)}"
            return JsonResponse(response_data)

        # Define paths
        raw_files_dir = RAW_FILES_DIR
        cleaned_files_dir = CLEANED_FILES_DIR

        # Create directories if they don't exist
        raw_files_dir.mkdir(parents=True, exist_ok=True)
        cleaned_files_dir.mkdir(parents=True, exist_ok=True)
```

```
5193            if processing_option == 'overwrite':
5194                logger.info("Overwrite option selected - removing existing files")
5195                # Remove all files in raw_files directory
5196                for file_path in raw_files_dir.glob('*.csv'):
5197                    try:
5198                        file_path.unlink()
5199                    except Exception as e:
5200                        logger.error(f"Error deleting raw file {file_path}: {str(e)}")
5201
5202                # Remove all files in cleaned_files directory
5203                for file_path in cleaned_files_dir.glob('*.csv'):
5204                    try:
5205                        file_path.unlink()
5206                    except Exception as e:
5207                        logger.error(f"Error deleting cleaned file {file_path}: {str(e)}")
5208
5209                # Remove index file if it exists
5210                index_file_path = BASE_DIR / "hesa_files_index.json"
5211                if index_file_path.exists():
5212                    try:
5213                        index_file_path.unlink()
5214                        logger.info("Removed existing index file")
5215                    except Exception as e:
5216                        logger.error(f"Error deleting index file: {str(e)}")
5217            # For 'add' option, clean only cleaned files and index, but keep raw files
5218            elif processing_option == 'add':
5219                logger.info("Add files option selected - cleaning only processed files")
5220                # Remove all files in cleaned_files directory
5221                for file_path in cleaned_files_dir.glob('*.csv'):
5222                    try:
5223                        file_path.unlink()
5224                        logger.info(f"Removed cleaned file: {file_path.name}")
5225                    except Exception as e:
5226                        logger.error(f"Error deleting cleaned file {file_path}: {str(e)}")
5227
5228                # Remove index file if it exists
5229                index_file_path = BASE_DIR / "hesa_files_index.json"
5230                if index_file_path.exists():
5231                    try:
5232                        index_file_path.unlink()
5233                        logger.info("Removed existing index file")
5234                    except Exception as e:
5235                        logger.error(f"Error deleting index file: {str(e)}")
5236
5237            # Process each uploaded file
5238            for file in files:
5239                file_name = file.name
5240                logger.info(f"Processing file: {file_name}")
5241
5242                # Check line count
5243                try:
5244                    # Read and count lines without loading entire file into memory
5245                    line_count = 0
5246                    for line in file.chunks():
5247                        line_count += line.count(b'\n')
5248
5249                    # Reset file pointer to beginning
```

```
5262            # Save file to raw_files directory
5263            try:
5264                destination_path = raw_files_dir / file_name
5265                with open(destination_path, 'wb+') as destination:
5266                    for chunk in file.chunks():
5267                        destination.write(chunk)
5268
5269                response_data['added_files'].append(file_name)
5270                logger.info(f"Successfully saved {file_name} to raw files directory")
5271            except Exception as e:
5272                logger.error(f"Error saving {file_name}: {str(e)}")
5273                response_data['error_files'].append(f"{file_name} (error saving file)")
5274                continue
5275
5276        # Run CSV cleaning script if files were added
5277        if response_data['added_files']:
5278            response_data['added_files_count'] = len(response_data['added_files'])
5279
5280            try:
5281                # Initialize the CSV processor
5282                from core.data_processing.csv_processor import CSVProcessor
5283                processor = CSVProcessor()
5284
5285                # Process the files
5286                results = processor.process_all_files()
5304                # Add the project root to the path if it's not already there
5305                project_root = str(BASE_DIR)
5306                if project_root not in sys.path:
5307                    sys.path.insert(0, project_root)
5308
5309                # Import create_index_file
5310                from csv_cleaning import create_index_file
5311
5312                # Generate the index file
5313                index_created = create_index_file(processor, results)
5314                response_data['indexing_created'] = index_created
5315
5316                # Count total files in the system
5317                response_data['total_files'] = len(list(raw_files_dir.glob('*.csv')))
5318
5319                # Set success flag
5320                response_data['success'] = True
5321
5322            except Exception as e:
5323                logger.error(f"Error during CSV cleaning process: {str(e)}")
5324                response_data['error'] = f"Error during processing: {str(e)}"
5325                response_data['cleaning_status'] = "Failed"
5326        else:
5327            response_data['error'] = "No valid files were uploaded"
5328    except Exception as e:
5329        logger.error(f"Unexpected error in file upload processing: {str(e)}")
5330        response_data['error'] = f"Unexpected error: {str(e)}"
5331    return JsonResponse(response_data)
```

D.3 Metadata Extraction

```
75      def extract_metadata(self, file_path: Path) -> Dict[str, Any]:      You, 2 months ago • change the way data is being p
76          """Extract metadata from the first line of the file if it exists."""
77          try:
78              with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
79                  first_line = f.readline().strip()
80
81              if first_line.startswith('#METADATA:'):
82                  # Extract the JSON metadata
83                  metadata_json = first_line[9:]  # Skip the #METADATA: prefix
84                  metadata = json.loads(metadata_json)
85                  return metadata
86              else:
87                  return {}
88          except Exception as e:
89              logger.warning(f"Error extracting metadata from {file_path}: {str(e)}")
90              return {}
```

D.4 Indexing

```python
def create_index_file(processor, results):      You, 2 months ago • created indeing file for the AI approch. fix th…
    """
    Creates an index file at the root level containing metadata for all CSV files
    in the cleaned_files directory.
    """
    logger = logging.getLogger(__name__)

    # Define path for the index file
    base_dir = Path(__file__).resolve().parent
    index_file_path = base_dir / "hesa_files_index.json"

    # Initialize index data structure
    index_data = {
        "hesa_files": [],
        "updated_at": datetime.datetime.now().isoformat(),
        "total_files": 0
    }

    # Process each cleaned file to extract metadata
    for file_name in processor.get_available_datasets():
        if file_name == '.gitkeep':
            continue

        cleaned_file_path = processor.clean_dir / file_name

        try:
            # Extract metadata from the cleaned file using the processor's method
            metadata = processor.extract_metadata(cleaned_file_path)

            if not metadata:
                logger.warning(f"No metadata found using processor for {file_name}, trying direct extraction")
                try:
                    # Read the first line to extract metadata
                    with open(cleaned_file_path, 'r', encoding='utf-8', errors='ignore') as f:
                        first_line = f.readline().strip()

                    if first_line.startswith('#METADATA:'):
                        # Extract the JSON metadata
                        metadata_json = first_line[9:]  # Skip the #METADATA: prefix
                        metadata = json.loads(metadata_json)
                        logger.info(f"Successfully extracted metadata directly from file: {file_name}")
                    else:
                        logger.warning(f"File {file_name} doesn't have metadata line")
                        metadata = {}
                except Exception as e:
                    logger.warning(f"Error extracting metadata from first line of {file_name}: {str(e)}")
                    metadata = {}

            # If still no metadata or missing key fields, try to create minimal metadata
            if not metadata or 'title' not in metadata or 'academic_year' not in metadata:
                logger.warning(f"Incomplete metadata for {file_name}, creating minimal metadata")

                # If title is missing, extract it from the raw file
                if 'title' not in metadata:
                    # Try to find the corresponding raw file
                    raw_file_path = processor.raw_dir / file_name

```

```
                    if raw_file_path.exists():
                        try:
                            # Extract title using the same logic as in processor.process_all_files
                            with open(raw_file_path, 'r', encoding='utf-8', errors='ignore') as f:
                                lines = [line.strip() for line in f.readlines()[:20]]

                            # Extract title from the first line
                            first_line = lines[0] if lines else ""

                            if 'Title:' in first_line:
                                # Get everything after 'Title:'
                                title_text = first_line.split('Title:')[1].strip()

                                # Remove leading commas and quotes
                                title_text = title_text.lstrip(',').strip('"')

                                # Find the part after "Table XX - " if it exists
                                import re
                                table_match = re.search(r'Table\s+\d+\s*-\s*(.*)', title_text)
                                if table_match:
                                    # Extract just the part after the dash
                                    extracted_title = table_match.group(1).strip()
                                    # Remove any trailing quotes
                                    extracted_title = extracted_title.rstrip('"')
                                    metadata['title'] = extracted_title
                                else:
                                    # If no table pattern, just use the whole text
                                    cleaned_title = title_text.strip()
                                    metadata['title'] = cleaned_title

                                logger.info(f"Extracted title from raw file: {metadata['title']}")
                            else:
                                # Fallback to cleaned filename
                                metadata['title'] = file_name.replace('.csv', '').split(' 20')[0]
                                logger.warning(f"Using fallback title from filename: {metadata['title']}")
                        except Exception as e:
                            logger.warning(f"Error extracting title from raw file: {str(e)}")
                            # Fallback to cleaned filename
                            metadata['title'] = file_name.replace('.csv', '').split(' 20')[0]
                    else:
                        # Fallback to cleaned filename
                        metadata['title'] = file_name.replace('.csv', '').split(' 20')[0]
                        logger.warning(f"Raw file not found, using fallback title from filename: {metadata['title']}")

                # If academic year is missing, extract it from the raw file first, then try filename
                if 'academic_year' not in metadata:
                    raw_file_path = processor.raw_dir / file_name

                    if raw_file_path.exists():
                        try:
                            # Read the first 20 lines to look for academic year
                            with open(raw_file_path, 'r', encoding='utf-8', errors='ignore') as f:
                                lines = [line.strip() for line in f.readlines()[:20]]

                            # Look for the exact pattern with comma at the start: ",Academic year,YYYY/YY"
                            academic_year_found = False
                            for line in lines:
                                # Skip lines with "to" as they indicate a range of years
```

```
                            if "to" in line:
                                continue

                            # Skip lines with "Subtitle:" that might contain "Academic years" text
                            if "Subtitle:" in line:
                                continue

                            # Look for patterns like ",Academic year,YYYY/YY" or "Filters:,Academic year,YYYY/YY"
                            if re.search(r'(?:,|\:)\s*Academic year\s*,\s*(20\d{2}/\d{2})', line):
                                year_match = re.search(r'(?:,|\:)\s*Academic year\s*,\s*(20\d{2}/\d{2})', line)
                                metadata['academic_year'] = year_match.group(1).strip()
                                logger.info(f"Found academic year format in raw file: {metadata['academic_year']}")
                                academic_year_found = True
                                break

                        # If not found in raw file, try filename pattern
                        if not academic_year_found:
                            import re
                            # Check for common academic year patterns in filename
                            year_match = re.search(r'(\d{4})[\.\-&_](\d{2})', file_name)
                            if year_match:
                                metadata['academic_year'] = f"{year_match.group(1)}/{year_match.group(2)}"
                                logger.info(f"Extracted academic year from filename: {metadata['academic_year']}")
                            else:
                                metadata['academic_year'] = "Unknown"
                                logger.warning(f"Could not extract academic year from filename: {file_name}")
                    except Exception as e:
                        logger.warning(f"Error extracting academic year from raw file: {str(e)}")
                        # Fall back to filename pattern
                        import re
                        year_match = re.search(r'(\d{4})[\.\-&_](\d{2})', file_name)
                        if year_match:
                            metadata['academic_year'] = f"{year_match.group(1)}/{year_match.group(2)}"
                        else:
                            metadata['academic_year'] = "Unknown"
                else:
                    # Fall back to filename pattern
                    import re
                    year_match = re.search(r'(\d{4})[\.\-&_](\d{2})', file_name)
                    if year_match:
                        metadata['academic_year'] = f"{year_match.group(1)}/{year_match.group(2)}"
                    else:
                        metadata['academic_year'] = "Unknown"

            # If columns are missing, add minimal columns from the file
            if 'columns' not in metadata or not metadata['columns']:
                try:
                    import pandas as pd
                    # Skip the metadata line and read the header row
                    df = pd.read_csv(cleaned_file_path, nrows=1, skiprows=1)
                    metadata['columns'] = df.columns.tolist()
                except Exception as e:
                    logger.warning(f"Error reading columns from {file_name}: {str(e)}")
                    metadata['columns'] = []

        # Ensure all required fields are present
        if 'columns' not in metadata:
            metadata['columns'] = []
```

```
203
204             # Add reference (file name)
205             metadata['reference'] = file_name
206
207             # Add metadata to index
208             index_data["hesa_files"].append(metadata)
209             logger.info(f"Added metadata for {file_name} to index with title: '{metadata.get('title', 'No title')}'")
210             logger.info(f"Academic Year: {metadata.get('academic_year', 'Unknown')}")
211
212         except Exception as e:
213             logger.error(f"Error processing {file_name} for index: {str(e)}")
214
215     # Update total files count
216     index_data["total_files"] = len(index_data["hesa_files"])
217
218     # Write index file
219     try:
220         with open(index_file_path, 'w', encoding='utf-8') as f:
221             json.dump(index_data, f, indent=2)
222         logger.info(f"Successfully created index file at {index_file_path}")
223         logger.info(f"Total files indexed: {index_data['total_files']}")
224         return True
225     except Exception as e:
226         logger.error(f"Error writing index file: {str(e)}")
227         return False
```

D.5 Cleaning Process

```python
    def process_single_file(self, file_name: str) -> bool:      You, 2 months ago • using regular expression predified query i
        """
        Process a single file by name from the raw directory.
        """
        file_path = self.raw_dir / file_name

        try:
            logger.info(f"Processing single file: {file_name}")

            # Validate file
            if not self.validate_csv(file_path):
                return False

            # Create new empty metadata
            metadata = {}

            # Read the first 20 lines to extract title and academic year
            try:
                with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
                    lines = [line.strip() for line in f.readlines()[:20]]

                # Extract title from the first line only
                first_line = lines[0] if lines else ""

                # Log the raw line for debugging
                logger.info(f"Raw first line: [{first_line}]")

                # Check if the line contains the title pattern
                if 'Title:' in first_line:
                    # Get everything after 'Title:'
                    title_text = first_line.split('Title:')[1].strip()
                    logger.info(f"After split by 'Title:': [{title_text}]")

                    # Remove leading commas and quotes
                    title_text = title_text.lstrip(',').strip('"')
                    logger.info(f"After cleaning commas/quotes: [{title_text}]")

                    # Find the part after "Table XX - " if it exists
                    table_match = re.search(r'Table\s+\d+\s*-\s*(.*)', title_text)
                    if table_match:
                        # Extract just the part after the dash
                        extracted_title = table_match.group(1).strip()
                        # Remove any trailing quotes
                        extracted_title = extracted_title.rstrip('"')
                        metadata['title'] = extracted_title
                        logger.info(f"Extracted title: [{extracted_title}]")
                    else:
                        # If no table pattern, just use the whole text
                        cleaned_title = title_text.strip()
                        metadata['title'] = cleaned_title
                        logger.info(f"No table pattern found, using cleaned title: [{cleaned_title}]")
                else:
                    logger.warning(f"First line does not contain 'Title:': [{first_line}]")

                # Extract academic year
                for line in lines:
                    # Skip lines with "to" as they indicate a range of years (e.g., "Academic years 2020/21 to 2023/24")
```

```
                if "to" in line:
                    continue

                # Look for patterns like ",Academic year,YYYY/YY" or "Filters:,Academic year,YYYY/YY"
                year_match = re.search(r'(?:,|\:)\s*Academic year\s*,\s*(20\d{2}/\d{2})', line)
                if year_match:
                    metadata['academic_year'] = year_match.group(1).strip()
                    logger.info(f"Found academic year format: {metadata['academic_year']}")
                    break
        except Exception as e:
            logger.warning(f"Error extracting title/academic year: {str(e)}")

        # If title not found, use "Unknown title"
        if 'title' not in metadata:
            logger.warning(f"Could not extract title from file content")
            metadata['title'] = "Unknown title"

        # If academic year not found, use "Unknown"
        if 'academic_year' not in metadata:
            logger.warning(f"Could not extract academic year from file content")

            # Try to extract from filename before defaulting to Unknown
            year_match = re.search(r'(\d{4})[\.\-&_](\d{2})', file_name)
            if year_match:
                metadata['academic_year'] = f"{year_match.group(1)}/{year_match.group(2)}"
                logger.info(f"Extracted academic year from filename: {metadata['academic_year']}")
            else:
                metadata['academic_year'] = "Unknown"

        # Clean data
        cleaned_df = self.clean_csv(file_path)
        if cleaned_df is None:
            return False

        # Extract title-based keywords
        title_keywords = self.extract_keywords_from_title(metadata['title'])
        metadata['keywords_title'] = title_keywords

        # Extract column-based keywords
        column_keywords = self.extract_keywords_from_columns(cleaned_df.columns)
        metadata['keywords_columns'] = column_keywords

        # Generate new filename based on title and academic year
        original_filename = file_name
        new_filename = self.generate_clean_filename(metadata['title'], metadata['academic_year'], original_filename)

        # Store original filename in metadata
        metadata['original_filename'] = original_filename
        # Store new filename in metadata
        metadata['new_filename'] = new_filename

        # Create metadata line
        metadata_line = f"#METADATA:{json.dumps(metadata)}\n"

        # Save cleaned file with metadata using new filename
        cleaned_path = self.clean_dir / new_filename

        # Write metadata line first, then the data
```

```
            with open(cleaned_path, 'w', encoding='utf-8') as f:
                f.write(metadata_line)

            # Then append the dataframe
            cleaned_df.to_csv(cleaned_path, index=False, mode='a')

            logger.info(f"Successfully processed {file_name} with metadata")
            logger.info(f"Title: {metadata.get('title', 'Not found')}")
            logger.info(f"Academic Year: {metadata.get('academic_year', 'Not found')}")
            logger.info(f"New filename: {new_filename}")
            logger.info(f"Title keywords: {title_keywords}")
            logger.info(f"Column keywords: {column_keywords}")
            return True

        except Exception as e:
            logger.error(f"Error processing {file_name}: {str(e)}")
            return False
```

D.6 CSVProcessor

```python
class CSVProcessor:
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def __init__(self):
        self.raw_dir = RAW_FILES_DIR
        self.clean_dir = CLEANED_FILES_DIR

    Windsurf: Refactor | Explain | ×
    def validate_csv(self, file_path: Path) -> bool:      You, 3 months ago • changed tailwind CND to direct instalation and ...
        """
        Validates if the CSV file is properly formatted and contains expected data.
        """
        try:
            df = pd.read_csv(file_path)

            # Check if file is empty
            if df.empty:
                logger.error(f"File {file_path} is empty")
                return False

            # Check for minimum required columns (customize based on your HESA data structure)
            required_columns = ['institution_id', 'year']  # Add your required columns
            missing_columns = [col for col in required_columns if col not in df.columns]

            if missing_columns:
                logger.error(f"Missing required columns: {missing_columns}")
                return False

            return True

        except Exception as e:
            logger.error(f"Error validating {file_path}: {str(e)}")
            return False

    Windsurf: Refactor | Explain | ×
    def clean_csv(self, file_path: Path) -> Optional[pd.DataFrame]:
        """
        Cleans the CSV data by handling missing values, removing duplicates, etc.
        """
        try:
            df = pd.read_csv(file_path)

            # Remove duplicate rows
            df = df.drop_duplicates()

            # Handle missing values (customize based on your needs)
            df = df.fillna({
                'numeric_columns': 0,  # Fill numeric missing values with 0
                'string_columns': 'Unknown'  # Fill string missing values with 'Unknown'
            })

            # Convert date columns to datetime (if any)
            date_columns = []  # Add your date column names
            for col in date_columns:
                df[col] = pd.to_datetime(df[col], errors='coerce')

            return df

        except Exception as e:
```

```
71              logger.error(f"Error cleaning {file_path}: {str(e)}")
72              return None
73
        Windsurf: Refactor | Explain | ×
74      def process_all_files(self) -> Dict[str, bool]:
75          """
76          Process all CSV files in the raw directory and save cleaned versions.
77          """
78          results = {}
79
80          # Create cleaned directory if it doesn't exist
81          self.clean_dir.mkdir(parents=True, exist_ok=True)
82
83          for file_path in self.raw_dir.glob('*.csv'):
84              try:
85                  # Validate file
86                  if not self.validate_csv(file_path):
87                      results[file_path.name] = False
88                      continue
89
90                  # Clean data
91                  cleaned_df = self.clean_csv(file_path)
92                  if cleaned_df is None:
93                      results[file_path.name] = False
94                      continue
95
96                  # Save cleaned file
97                  cleaned_path = self.clean_dir / file_path.name
98                  cleaned_df.to_csv(cleaned_path, index=False)
99                   results[file_path.name] = True
100
101                  logger.info(f"Successfully processed {file_path.name}")
102
103              except Exception as e:
104                  logger.error(f"Error processing {file_path.name}: {str(e)}")
105                  results[file_path.name] = False
106
107         return results
108
        Windsurf: Refactor | Explain | ×
109     def get_available_datasets(self) -> List[str]:
110         """
111         Returns a list of available cleaned dataset names.
112         """
113         return [f.name for f in self.clean_dir.glob('*.csv')]
```

E: Query Processing and Interpretation

E.1 Transform free text into query parameters

```python
    def analyze_query(self, query: str) -> Dict[str, Any]:      You, 2 months ago • created indeing file for the AI approch.
        """
        Analyze a natural language query using the Gemini API.

        """
        import requests
        import json
        import re
        import logging

        logger = logging.getLogger(__name__)
        logger.info(f"Analyzing query with Gemini API: {query}")

        if not self.api_key:
            logger.error("No Gemini API key provided")
            raise ValueError("Gemini API key is required")

        # Construct the prompt
        system_prompt = """
        You are a helpful assistant that extracts structured information from a user's query about higher education statistics

        Extract the following information and output it in JSON format:
        1. Institutions mentioned (e.g., "University of Oxford", "University of Leicester")
        2. Years mentioned (e.g., "2020", "2020/21")
        3. If a year range is given, identify the start_year and end_year
        4. The type of data being requested (e.g., "student numbers", "enrollment", "graduates")

        IMPORTANT: You need to detect and correct ONLY SPELLING typos in institution names and years.
        For institutions, follow these strict rules:
        - ONLY consider a term to have a typo if it has clear misspellings (e.g., "Univercity" → "University", "Liecester" → "Leicester")
        - DO NOT consider "Oxford University" vs "University of Oxford" as a typo - these are different ways to refer to the same institution
        - DO NOT mark an institution as having a typo if it's spelled correctly but uses a different naming convention
        - DO NOT transform a city name like "london" into "The University of London"
        - PRESERVE the original terms from the query - if user says "london", keep it as "london" in the institutions list
        - If there are no spelling errors, original_institutions should exactly match institutions, and has_institution_typos should be false

        For years:
        - ONLY mark years as having typos if they have clear numerical errors (e.g., "20025" → "2025")
        - If no year typos exist, original_years should exactly match years, and has_year_typos should be false

        Be careful about interpreting years in academic context:
        - If the query contains phrases like "starting in [YEAR]" or "beginning in [YEAR]", interpret [YEAR] as the start of an academic year.
        - If the query contains phrases like "end of [YEAR]" or "ending in [YEAR]", interpret [YEAR] as the end of an academic year.
        - For ranges like "2016 to 2017", treat both as starting years of academic years.
        - If there's no clarification, assume a year refers to the starting year of an academic year.
        - For "past X years", calculate the years based on the current year (2025).

        For the data_request field, use the SPECIFIC TERMS from the query whenever possible:
        - If the query mentions "undergraduates", use ["undergraduate"] not ["student_enrollment"]
        - If the query mentions "postgraduates", use ["postgraduate"] not ["student_enrollment"]
        - If the query mentions "student numbers", use ["student_count"]      You, last month • updated the way Geminin finds
```

```
    - If the query mentions "enrollment" or "enrolment", use ["enrollment"]
    - If the query mentions "staff" or "teachers", use ["staff_data"]
    - If the query mentions "research", use ["research_data"]
    - NEVER generalize specific educational levels - keep them exactly as requested

    Output format:
    {
      "institutions": ["University X", "london"],
      "original_institutions": ["Universcity X", "london"],
      "has_institution_typos": true,
      "years": ["2019/20", "2020/21"],
      "original_years": ["20019", "2020"],
      "has_year_typos": true,
      "start_year": "2019",
      "end_year": "2020",
      "data_request": ["undergraduate", "graduation_rates"]
    }

    If no specific institutions are mentioned, return an empty list for institutions and original_institutions.
    If no typos were detected, original_institutions should match institutions, and has_institution_typos should be false.
    The same applies to years and original_years.
    """

    # The actual user query
    user_prompt = f"Extract information from this query, paying special attention to academic year conventions and
    potential typos in institution names and years: '{query}'"

    try:
        # Gemini API endpoint - Updated to use the most current endpoint
        url = "https://generativelanguage.googleapis.com/v1/models/gemini-1.5-pro:generateContent"

        # Request payload
        payload = {
            "contents": [
                {
                    "role": "user",
                    "parts": [
                        {"text": system_prompt},
                        {"text": user_prompt}
                    ]
                }
            ],
            "generationConfig": {
                "temperature": 0.1,
                "topP": 0.95,
                "topK": 40,
                "maxOutputTokens": 2048
            }
        }

        # Headers with API key
        headers = {
            "Content-Type": "application/json",
            "x-goog-api-key": self.api_key
        }
```

```
        # Make the request
        response = requests.post(url, json=payload, headers=headers)

        # Check for successful response
        if response.status_code != 200:
            logger.error(f"Gemini API error: {response.status_code} - {response.text}")
            raise Exception(f"Gemini API returned status code {response.status_code}: {response.text}")

        # Parse the response
        response_data = response.json()

        # Extract text from response
        if 'candidates' in response_data and len(response_data['candidates']) > 0:
            if 'content' in response_data['candidates'][0]:
                response_text = response_data['candidates'][0]['content']['parts'][0]['text']
            else:
                logger.error("No content in Gemini API response")
                raise Exception("No content in Gemini API response")
        else:
            logger.error("No candidates in Gemini API response")
            raise Exception("No candidates in Gemini API response")

        # Find the JSON response within the text
        json_match = re.search(r'\{.*\}', response_text, re.DOTALL)
        if not json_match:
            logger.error(f"Failed to extract JSON from response: {response_text}")
            raise Exception("Failed to extract JSON from the Gemini API response")

```

```
170
171             # Parse the JSON data
172             result = json.loads(json_match.group(0))
173
174             # Default values for missing fields
175             result.setdefault('institutions', [])
176             result.setdefault('original_institutions', [])
177             result.setdefault('has_institution_typos', False)
178             result.setdefault('years', [])
179             result.setdefault('original_years', [])
180             result.setdefault('has_year_typos', False)
181             result.setdefault('data_request', ['general_data'])
182
183             # Always ensure University of Leicester is included if any institution is mentioned
184             if result['institutions'] and 'University of Leicester' not in result['institutions']:
185                 result['institutions'].append('University of Leicester')
186
187             # Fix "in University" issue by filtering out partial names
188             result['institutions'] = [
189                 inst for inst in result['institutions']
190                 if len(inst.split()) > 1  # Must be at least two words
191                 and inst.lower() != "in university"  # Explicitly exclude "in university"
192             ]
193
194             # If we don't have typo information but have institutions, assume no typos
195             if 'original_institutions' not in result or not result['original_institutions']:
196                 result['original_institutions'] = result['institutions'].copy()
197                 result['has_institution_typos'] = False
200             if 'original_years' not in result or not result['original_years']:
201                 result['original_years'] = result['years'].copy()
202                 result['has_year_typos'] = False
203
204             # Handle special case for "past X years"
205             if 'start_year' not in result or 'end_year' not in result:
206                 # Check if the query contains "past X years"
207                 past_years_match = re.search(r'past\s+(\d+)\s+years?', query.lower())
208                 if past_years_match:
209                     num_years = int(past_years_match.group(1))
210                     # Use current year from datetime
211                     current_year = datetime.now().year
212                     end_year = current_year
213                     start_year = current_year - num_years
214
215                     result['end_year'] = str(end_year)
216                     result['start_year'] = str(start_year)
217
218                     # Expand years array to include all years in the range
219                     if not result['years']:
220                         for year in range(start_year, end_year + 1):
221                             # Only add academic year format, not plain year
222                             result['years'].append(f"{year}/{str(year+1)[2:4]}")
223
224             # Apply our academic year logic for special cases if Gemini didn't handle it well
225             self._process_academic_year_logic(query, result)
226
227             logger.info(f"Extracted information: {result}")
228             return result
```

E.2 Date Interpretation

```python
    def _process_academic_year_logic(self, query, result):      You, 2 months ago • changed the way dataset is created. it now
        """
        Process academic year logic based on context clues in the query.
        This handles cases like:
        - "starting in 2017" → 2017/18
        - "end of 2017" → 2016/17
        - "2016 to 2017" → 2016/17 - 2017/18 (both as starting years)

        """
        import re
        import logging

        logger = logging.getLogger(__name__)
        query_lower = query.lower()

        # Extract years with context
        start_patterns = [
            r'start(?:ing|s|ed)?\s+(?:in|from|at)?\s+(?:the\s+)?(?:year\s+)?(\d{4})',
            r'begin(?:ning|s)?\s+(?:in|from|at)?\s+(?:the\s+)?(?:year\s+)?(\d{4})',
            r'from\s+(?:the\s+)?(?:year\s+)?(\d{4})'
        ]

        end_patterns = [
            r'end(?:ing|s|ed)?\s+(?:in|at|of)?\s+(?:the\s+)?(?:year\s+)?(\d{4})',
            r'finish(?:ing|es|ed)?\s+(?:in|at)?\s+(?:the\s+)?(?:year\s+)?(\d{4})',
            r'(?:in|at)\s+(?:the\s+)?end\s+of\s+(?:the\s+)?(?:year\s+)?(\d{4})'
        ]

        # Handle range logic for years without context
        if 'start_year' in result and 'end_year' in result and result['start_year'] is not None and result['end_year'] is not
        None:
            # Range years should be treated as starting years of academic years
            start_year = result['start_year']
            end_year = result['end_year']

            try:
                # Clear existing years array to rebuild it with correct academic years
                result['years'] = []

                # Add academic years for the range
                for year in range(int(start_year), int(end_year) + 1):
                    academic_year = f"{year}/{str(year+1)[2:4]}"
                    result['years'].append(academic_year)

                logger.info(f"Processed year range: {start_year}-{end_year} as academic years: {', '.join(result['years'])}")
            except (ValueError, TypeError) as e:
                logger.error(f"Error processing year range: {e}. start_year={start_year}, end_year={end_year}")

            return

        # Explicitly defined start years
        for pattern in start_patterns:
            start_match = re.search(pattern, query_lower)
            if start_match:
                year = start_match.group(1)
                try:
                    academic_year = f"{year}/{str(int(year)+1)[2:4]}"
```

```
                    if academic_year not in result['years']:
                        result['years'].append(academic_year)
                        result['start_year'] = year
                        logger.info(f"Processed starting year {year} as academic year {academic_year}")
                except (ValueError, TypeError) as e:
                    logger.error(f"Error processing starting year {year}: {e}")

        # Explicitly defined end years
        for pattern in end_patterns:
            end_match = re.search(pattern, query_lower)
            if end_match:
                year = end_match.group(1)
                try:
                    previous_year = str(int(year) - 1)
                    academic_year = f"{previous_year}/{year[2:4]}"

                    if academic_year not in result['years']:
                        result['years'].append(academic_year)
                        result['end_year'] = year
                        logger.info(f"Processed ending year {year} as academic year {academic_year}")
                except (ValueError, TypeError) as e:
                    logger.error(f"Error processing ending year {year}: {e}")

        # For single years without context, treat as starting years
        plain_year_pattern = r'\b(20\d{2})\b'
        years_mentioned = re.findall(plain_year_pattern, query)

        # Skip years that were already processed with context
                    already_processed = True
                    break

            if not already_processed:
                try:
                    # No context, treat as starting year
                    academic_year = f"{year}/{str(int(year)+1)[2:4]}"
                    if academic_year not in result['years']:
                        result['years'].append(academic_year)
                        # If no explicit start_year is set, use this as default
                        if 'start_year' not in result or result['start_year'] is None:
                            result['start_year'] = year
                        logger.info(f"Processed plain year {year} as academic year {academic_year}")

                        # Remove the plain year from the years list if it exists
                        if year in result['years']:
                            result['years'].remove(year)
                            logger.info(f"Removed plain year {year} from years list to avoid duplication")
                except (ValueError, TypeError) as e:
                    logger.error(f"Error processing plain year {year}: {e}")

        # Make sure years are unique
        result['years'] = list(set(result['years']))

        # Filter out any plain years that might still be in the list from other sources
        result['years'] = [year for year in result['years'] if '/' in year]
```

F: Visualization Generation

F.1 Chart Generation

```python
def generate_visualization(client, dataset_info, user_request, chart_type=None):
    """
    Generate a Chart.js visualization based on the dataset and user request
    """
    try:
        # Extract relevant information from dataset_info
        title = dataset_info.get('title', '')
        columns = dataset_info.get('columns', [])
        rows = dataset_info.get('rows', [])
        query = dataset_info.get('query', '')
        institutions = dataset_info.get('institutions', [])
        years = dataset_info.get('years', [])
        all_year_data = dataset_info.get('allYearData', [])

        # Log information about the request
        logging.info(f"Visualization request for dataset: {title}")
        logging.info(f"User request: {user_request}")
        logging.info(f"Years detected: {years}")
        logging.info(f"Number of rows: {len(rows)}")
        logging.info(f"Number of all year data rows: {len(all_year_data) if all_year_data else 0}")
        logging.info(f"Requested chart type: {chart_type}")

        # Extract institution names directly from the dataset
        dataset_institutions = []
        provider_col_idx = -1
```

```python
        if chart_type == 'line' and not dataset_characteristics["has_multiple_years"]:
            chart_compatibility_check = """
            IMPORTANT NOTE: The user has requested a line chart, but this dataset only contains data for a single academic year.
            Line charts are generally better suited for showing trends over multiple time periods.

            You should:
            1. Try to generate a line chart as requested, adapting the data to work with a single year if possible
            2. Make sure the chart configuration is valid JavaScript that can be evaluated
            3. Explicitly mention the limitation of using a line chart with single-year data in your insights
            """
            chart_compatibility_issue = True

        elif chart_type == 'pie' and len([col for col in columns if col not in ['HE provider', 'Academic Year']]) > 5:
            chart_compatibility_check = """
            IMPORTANT NOTE: The user has requested a pie chart with a dataset that has many data columns.
            Pie charts work best with a small number of categories (typically 5 or fewer).

            You should:
            1. Generate a pie chart as requested, but focus on the most important categories      You, last month • so progre
            2. Make sure the chart configuration is valid JavaScript that can be evaluated
            3. Mention the limitation of using pie charts with many categories in your insights
            """
            chart_compatibility_issue = True
```

```
        if len(institutions) > 5 and 'compare' in user_request.lower() and any(inst.lower() in user_request.lower() for inst
        in institutions):
            too_many_institutions_warning = """
            IMPORTANT NOTE: The user has requested comparing multiple institutions, but displaying too many institutions
            on a single chart can make it cluttered and hard to read.

            You should:
            1. Limit the visualization to the 3-5 most relevant institutions mentioned in the user's request
            2. If the user explicitly mentioned specific institutions, prioritize those
            3. Mention in the insights that you've limited the chart to the most relevant institutions for readability
            """
            chart_compatibility_check += too_many_institutions_warning
            too_many_institutions = True
            chart_compatibility_issue = True

        # Ensure 'The University of Leicester' is always considered
        leicester_variants = ['university of leicester', 'the university of leicester', 'leicester university']
        leicester_included = False
        for inst in institutions:
            if inst.lower() in leicester_variants:
                leicester_included = True
                break

        if not leicester_included and 'leicester' not in ' '.join(institutions).lower():
            institutions.append('The University of Leicester')
            logging.info("Added University of Leicester to institutions list")

        institution_analysis = ""
        if len(mentioned_institutions) > 0 and len(referenced_institutions) == 0:
            # User mentioned institutions but none match our dataset
            institution_analysis = f"""
            IMPORTANT WARNING: The user has requested information about institutions that don't exist in the dataset.
            The available institutions in this dataset are: {', '.join(institutions[:10])}{'...' if len(institutions) > 10
            else ''}

            You MUST adapt the visualization to use only institutions that exist in the dataset and clearly explain in the
            insights
            that the requested institution(s) aren't available in this dataset.
            """
        elif any(university_term in user_request.lower() for university_term in ["university", "college", "school"]) and len
        (referenced_institutions) == 0:
            # Generic institution terms but no specific matches
            institution_analysis = """
            IMPORTANT WARNING: The user's request mentions educational institutions, but I couldn't determine which specific
            institutions from the dataset they're interested in.

            Please create a visualization using the institutions available in the dataset, and clearly explain in the
            insights
            which institutions were selected and why.
            """

```

```
4056        axis_instructions = ""
4057        if "axis" in user_request.lower() or "axes" in user_request.lower():
4058            axis_instructions = """
4059            I've noticed the user has specified some axis preferences. Please follow these instructions carefully:
4060            1. If the user wants to swap X and Y axes, make sure to implement this exactly as requested
4061            2. If the user specifies what should be on X or Y axis, honor this request precisely
4062            3. For pie charts, note that traditional X/Y axes don't apply, but use the specified dimensions for data
                selection
4063            """
4064
4065        # Add any compatibility warnings for the selected chart type
4066        chart_compatibility_note = ""
4067        if chart_type:
4068            chart_compatibility_warning = get_chart_type_explanation(chart_type, dataset_characteristics)
4069            if chart_compatibility_warning:
4070                chart_compatibility_note = f"\n\nIMPORTANT NOTE ABOUT {chart_type.upper()} CHARTS:
                    {chart_compatibility_warning}"
4071
4072        # Set instructions for requested chart type
4073        chart_type_instruction = ""
4074        if chart_type:
4075            chart_type_instruction = f"""
4076            IMPORTANT: You MUST generate a {chart_type} chart for this visualization. Do not suggest or use any other chart
                type.
4077            The user has specifically requested a {chart_type} chart, so optimize the visualization for this chart type.
4078            {chart_compatibility_check}
4079
4073        chart_type_instruction = ""
4074        if chart_type:
4075            chart_type_instruction = f"""
4076            IMPORTANT: You MUST generate a {chart_type} chart for this visualization. Do not suggest or use any other chart
                type.
4077            The user has specifically requested a {chart_type} chart, so optimize the visualization for this chart type.
4078            {chart_compatibility_check}
4079
4080            For a {chart_type} chart specifically:
4081            - If it's a 'pie' chart: Ensure the Chart.js configuration has type: 'pie' and data formatted for a pie chart
4082            - If it's a 'line' chart: Ensure the Chart.js configuration has type: 'line' and datasets with x/y coordinates
4083            - If it's a 'bar' chart: Ensure the Chart.js configuration has type: 'bar' with labels and dataset values
4084
4085            CRITICAL: The JSON must be valid JavaScript that can be evaluated directly.
4086            DO NOT use string literals with quotes, Object.assign, or template literals in the chart configuration.
4087            """
4088
4089        # Create years description
4090        years_description = f"{len(years)} academic year{'s' if len(years) > 1 else ''}"
4091
```

```
        IMPORTANT INSTRUCTIONS:
        1. Generate a Chart.js configuration for a {chart_type if chart_type else "suitable"} chart visualization.
        2. STRICTLY USE ONLY the {chart_type.upper() if chart_type else "appropriate"} CHART TYPE for visualization, even if another type might be better suited.
        3. Ensure the visualization accurately represents the data and addresses the user's request.
        4. ONLY use the exact institution names as they appear in the dataset.
        5. If the user mentions institutions not in the dataset, adapt the visualization using only available institutions and clearly explain this in the insights.
        6. Generate clear and insightful analysis of the visualization.
        7. If the user's request cannot be fulfilled with the available data, provide a clear explanation.
        8. Make sure colors are distinct and the visualization is easy to interpret.
        9. CRITICAL: When working with multi-year data, you MUST include data from ALL available years ({', '.join(years)}) in your visualization and analysis.
        10. Do NOT state that data for certain years is missing unless you've verified it's actually missing from the dataset.
        11. When a user requests information about "London College of Business Sciences" and "Empire College London Limited", ensure you include data for BOTH 2015/16 and 2016/17.
        12. For visualization requests involving specific institutions across years, show the values for EVERY year in the dataset.

        CRITICAL FOR TIME-BASED VISUALIZATIONS:
        1. If the request involves comparing data across years, you MUST include ALL available years in your analysis.
        2. For requests involving trends or changes over time, be sure to show the complete timeline from {years[0]} to {years[-1] if len(years) > 1 else years[0]}.
        3. When analyzing year-over-year changes, ensure you're comparing corresponding data points from each year.
        4. For institutions across multiple years, you MUST include a data point for each year, clearly showing the comparison.
        5. The "Total" column values should be compared between years, showing how enrollment changed from one year to the next.
```

F.2 Handles different chart type requests

```python
def get_chart_data(request, chart_type):      You, 2 months ago • created charts of type pie, line and graph for …
    try:
        # Path to the CSV file
        csv_path = os.path.join(settings.BASE_DIR, 'data', 'cleaned_files', 'chart-1_cleaned.csv')

        # Read the CSV file with pandas
        try:
            # First read the metadata to determine where data starts
            with open(csv_path, 'r') as f:
                lines = f.readlines()

            # Find where the actual data starts (after metadata)
            data_start = 0
            for i, line in enumerate(lines):
                if 'Academic Year' in line or 'Level of study' in line:
                    data_start = i
                    break

            # Read the CSV file starting from the data
            df = pd.read_csv(csv_path, skiprows=data_start)

            # Clean up column names
            df.columns = df.columns.str.strip()

            # Ensure required columns exist
            if 'Level of study' not in df.columns:
                raise ValueError("Column 'Level of study' not found in the data")
            if 'Academic Year' not in df.columns:
                raise ValueError("Column 'Academic Year' not found in the data")
            if 'Number' not in df.columns:
                raise ValueError("Column 'Number' not found in the data")

            # Convert Number to numeric, replacing any non-numeric values with NaN
            df['Number'] = pd.to_numeric(df['Number'], errors='coerce')

            # Drop any rows with missing values
            df = df.dropna(subset=['Number', 'Level of study', 'Academic Year'])

        except Exception as e:
            raise ValueError(f"Error reading CSV file: {str(e)}")

        # Create figure (using Agg backend)
        plt.figure(figsize=(10, 6))

        if chart_type == 'line':
            # Group by years and plot each level of study
            for level in df['Level of study'].unique():
                level_data = df[df['Level of study'] == level]
                plt.plot(level_data['Academic Year'], level_data['Number'], label=level, marker='o')
            plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
            plt.xticks(rotation=45)
            plt.title('Students by Level of Study Over Time')

        elif chart_type == 'bar':
            # Group by Level of study and sum the numbers
            data = df.groupby('Level of study')['Number'].sum().sort_values(ascending=False)
            plt.bar(range(len(data)), data.values)
```

```
398         else:
399             raise ValueError(f"Unsupported chart type: {chart_type}")
400
401         # Adjust layout
402         plt.tight_layout()
403
404         # Save to bytes buffer
405         buffer = io.BytesIO()
406         plt.savefig(buffer, format='png', bbox_inches='tight', dpi=300)
407         buffer.seek(0)
408         image_png = buffer.getvalue()
409         buffer.close()
410
411
412         # Encode
413         graphic = base64.b64encode(image_png)
414         graphic = graphic.decode('utf-8')
415
416         return CustomJsonResponse({
417             'data': {
418                 'image': f'{graphic}'
419             }
420         })
421
422     except Exception as e:
423         return CustomJsonResponse({'error': str(e)}, status=500)
424
```

rt/<str:chart_type>/', views.get_chart_data, name='get_chart_data'),

F.3 Gets AI recommendations for chart types

```
3522  def get_chart_recommendation(client, dataset_info):      You, last month • For the selected dataset Gemini gives recommend…
3523      """
3524      Get a chart type recommendation from Gemini based on the dataset
3525      """
3526      try:
3527          # Extract relevant information from dataset_info
3528          title = dataset_info.get('title', '')
3529          columns = dataset_info.get('columns', [])
3530          sample_rows = dataset_info.get('rows', [])[:5]  # Use up to 5 rows as a sample
3531          query = dataset_info.get('query', '')
3532          institutions = dataset_info.get('institutions', [])
3533          years = dataset_info.get('years', [])
3534          all_year_data = dataset_info.get('allYearData', [])
3535
3536          # Log information about the data
3537          logging.info(f"Chart recommendation for dataset: {title}")
3538          logging.info(f"Years detected: {years}")
3539          logging.info(f"Number of institutions: {len(institutions)}")
3540          logging.info(f"Institutions: {institutions}")
3541
3542          # Extract institution names directly from the dataset
3543          dataset_institutions = []
3544          if sample_rows and len(sample_rows) > 0:
3545              # Find the institution/provider column index
3546              provider_col_idx = -1
3547              for i, col in enumerate(columns):
3548                  if col.lower() in ['he provider', 'institution', 'provider', 'university']:
3597          prompt = f"""
3598          I have a dataset titled "{title}" with the following columns:
3599          {', '.join(columns)}
3600
3601          Here are some sample rows from the dataset:
3602          {format_sample_rows(columns, sample_rows)}
3603
3604          The institutions in this dataset are: {', '.join(institutions)}
3605          The academic years in the dataset are: {', '.join(years)}
3606
3607          IMPORTANT INSTRUCTIONS:
3608          1. Analyze this specific dataset and determine the most appropriate chart type based solely on the data
              characteristics.
3609          2. ONLY use the EXACT institution names from the list I provided - do not modify, abbreviate or combine them.
3610          3. Do not mention or suggest institutions that aren't in the provided dataset list.
3611          4. Consider the following chart theory principles:
3612             - Line charts are best for showing trends and changes over time (multiple years/time periods)
3613             - Bar charts are best for comparing data across different categories or groups
3614             - Pie charts are best for showing proportions of parts to a whole (typically at a single point in time)
3615          5. Ignore the user's original query completely and focus only on what visualizations would be most valuable for this
              dataset.
3616
3617          Please provide:
3618          1. The recommended chart type (bar, line, or pie) that would be most appropriate for this specific dataset
3619          2. A clear explanation of why this chart type is recommended based on the dataset characteristics
3620          3. Three specific example visualization requests that:
3621             - Only use institutions and years that are available in this dataset
3622             - Must use the EXACT institution names as listed in the dataset - never abbreviate them
3623             - Would work well with the recommended chart type
3624             - Are realistic and can be fulfilled with the available data
```

F.4 Dynamically changing the visualization

```
4536  def change_chart_type(client, dataset_info, chart_type, original_recommendation, current_request=''):      You, last month •
4537      """
4538      Generate a new chart recommendation and examples for a different chart type
4539      """
4540      try:
4541          # Extract relevant information from dataset_info
4542          title = dataset_info.get('title', '')
4543          columns = dataset_info.get('columns', [])
4544          sample_rows = dataset_info.get('rows', [])[:5]  # Use up to 5 rows as a sample
4545          query = dataset_info.get('query', '')
4546          institutions = dataset_info.get('institutions', [])
4547          years = dataset_info.get('years', [])
4548          all_year_data = dataset_info.get('allYearData', [])
4549
4550          # Log information about the request
4551          logging.info(f"Chart type change request for dataset: {title}")
4552          logging.info(f"Changing to chart type: {chart_type}")
4553          logging.info(f"Original recommendation: {original_recommendation}")
4554          logging.info(f"Current visualization request: {current_request}")
4555
4631          # Special case checks for incompatible chart types
4632          if chart_type == 'pie' and current_request and ('axis' in current_request.lower() or 'axes' in current_request.lower
              () or 'swap' in current_request.lower()):
4633              compatibility_warning = "Pie charts don't have traditional X and Y axes, so axis swapping or specific axis
                  assignments don't apply to this chart type."
4634          elif chart_type == 'line' and not dataset_characteristics["has_multiple_years"] and 'year' in current_request.lower()
              :
4635              compatibility_warning = "Line charts work best with multiple time periods. Your request involves a time-based
                  comparison but the dataset might not have enough time periods to show meaningful trends."
4636
4637          # If we have a compatibility warning, include it in the response
4638          compatibility_note = ""
4639          if compatibility_warning:
4640              compatibility_note = f"<div class='text-amber-600 mb-2 font-medium'><strong>Note:</strong>
                  {compatibility_warning}</div>"
4641
4642          # Format information about multi-year data if available
4643          multi_year_info = ""
4644          if years and len(years) > 0:
4645              multi_year_info = f"\nThis dataset contains data for multiple academic years: {', '.join(years)}."
4646              multi_year_info += "\nConsider suggesting visualizations that could compare trends across these years or focus
                  on specific years."
4647
```

```
4690            prompt = f"""
4691            I have a dataset titled "{title}" with the following columns:
4692            {', '.join(columns)}
4693
4694            Here are some sample rows from the dataset:
4695            {format_sample_rows(columns, sample_rows)}
4696
4697            The institutions in this dataset are: {', '.join(institutions)}
4698            The academic years in the dataset are: {', '.join(years)}
4699
4700            IMPORTANT INSTRUCTIONS:
4701            1. I need examples SPECIFICALLY for a {chart_type.upper()} CHART. Every example MUST work with this chart type.
4702            2. ONLY use the EXACT institution names from the list I provided - do not modify, abbreviate or combine them.
4703            3. If a {chart_type} chart isn't ideal for some aspects of the data, still create examples that WILL WORK with a
                {chart_type} chart.
4704            4. Your examples will be used as-is and rendered as a {chart_type} chart, so they MUST be appropriate for this
                chart type.
4705            5. Focus on what {chart_type} charts do best:
4706               - LINE CHARTS: Show trends over time/years
4707               - BAR CHARTS: Compare values across categories/institutions
4708               - PIE CHARTS: Show proportions of a whole for a single period
4709
4710            Please provide:
4711            1. A brief explanation of how a {chart_type} chart could be used with this specific dataset, noting its
                strengths and limitations.
4712            2. 3 example visualization requests that:
4713               - Are SPECIFICALLY designed to work with a {chart_type} chart
4714               - ONLY use institutions and years that are actually available in this dataset (as listed above)
4715               - Are realistic and can be fulfilled with the available data
```


```
4690            prompt = f"""
4691            I have a dataset titled "{title}" with the following columns:
4692            {', '.join(columns)}
4693
4694            Here are some sample rows from the dataset:
4695            {format_sample_rows(columns, sample_rows)}
4696
4697            The institutions in this dataset are: {', '.join(institutions)}
4698            The academic years in the dataset are: {', '.join(years)}
4699
4700            IMPORTANT INSTRUCTIONS:
4701            1. I need examples SPECIFICALLY for a {chart_type.upper()} CHART. Every example MUST work with this chart type.
4702            2. ONLY use the EXACT institution names from the list I provided - do not modify, abbreviate or combine them.
4703            3. If a {chart_type} chart isn't ideal for some aspects of the data, still create examples that WILL WORK with a
                {chart_type} chart.
4704            4. Your examples will be used as-is and rendered as a {chart_type} chart, so they MUST be appropriate for this
                chart type.
4705            5. Focus on what {chart_type} charts do best:
4706               - LINE CHARTS: Show trends over time/years
4707               - BAR CHARTS: Compare values across categories/institutions
4708               - PIE CHARTS: Show proportions of a whole for a single period
4709
4710            Please provide:
4711            1. A brief explanation of how a {chart_type} chart could be used with this specific dataset, noting its
                strengths and limitations.
4712            2. 3 example visualization requests that:
4713               - Are SPECIFICALLY designed to work with a {chart_type} chart
4714               - ONLY use institutions and years that are actually available in this dataset (as listed above)
4715               - Are realistic and can be fulfilled with the available data
```

G: Front-End Rendering

G.1 Rendering Visualization

```
1332  function renderChart(chartConfig) {      You, last month • For the selected dataset Gemini gives recommend…
1333      // Get the canvas element for rendering
1334      const canvas = document.getElementById('visualizationChart');
1335      const ctx = canvas.getContext('2d');
1336
1337      try {
1338          // If we have a previous chart, destroy it properly
1339          if (window.currentChart) {
1340              window.currentChart.destroy();
1341              window.currentChart = null;
1342          }
1343
1344          // In case the chart wasn't properly destroyed, clear the canvas manually
1345          canvas.width = canvas.width; // This effectively clears the canvas
1346
1347          // Create a new chart with the provided configuration
1348          window.currentChart = new Chart(ctx, chartConfig);
1349
1350          // Ensure chart is visible
1351          document.getElementById('visualizationContainer').classList.remove('hidden');
1352          document.getElementById('visualizationError').classList.add('hidden');
1353
1354      } catch (error) {
1355          console.error("Error rendering chart:", error);
1356
1357          // Display error to user with friendly message based on chart type
1358          let errorMessage = "The chart couldn't be displayed with your current settings.";
1359
1360          if (chartConfig && chartConfig.type) {
1361              errorMessage = `Your ${chartConfig.type} chart couldn't be created with the selected data.`;
1362              errorMessage += "<br><br>Try selecting a different chart type or modifying your request.";
1363          }
1364
1365          document.getElementById('visualizationErrorMessage').innerHTML = errorMessage;
1366          document.getElementById('visualizationError').classList.remove('hidden');
1367          document.getElementById('visualizationContainer').classList.add('hidden');
1368      }
1369  }
```

G.2 Interaction Control

```
1112  function generateVisualization(datasetInfo, request, chartType) {      You, last month • so progress with charts but still n
1113      const loadingEl = document.getElementById('visualizationLoading');
1114      const errorEl = document.getElementById('visualizationError');
1115      const errorMsgEl = document.getElementById('visualizationErrorMessage');
1116      const alternativesEl = document.getElementById('alternativeSuggestions');
1117      const suggestionsListEl = document.getElementById('suggestionsList');
1118      const visualizationEl = document.getElementById('visualizationContainer');
1119      const insightsEl = document.getElementById('insightsContent');
1120
1121      // Store the current request for future use
1122      currentRequest = request;
1123
1124      // Hide previous results
1125      errorEl.classList.add('hidden');
1126      alternativesEl.classList.add('hidden');
1127      visualizationEl.classList.add('hidden');
1128
1129      // Show loading state
1130      loadingEl.classList.remove('hidden');
1131
1132      // Prepare data for API call
1133      const requestData = {
1134          action: 'generate_visualization',
1135          dataset_info: datasetInfo,
1136          user_request: request
1137      };
1138
1139      // If a specific chart type was requested, include it
1140      if (chartType) {
1141          requestData.chart_type = chartType;
1142
1143          // Set active chart button to reflect the selected type
1144          setActiveChartButton(chartType);
1145
1146          // Add a confirmation that this chart type is being enforced
1147          console.log(`Enforcing chart type: ${chartType} for visualization`);
1148
1149          // If the chart type wasn't specifically mentioned in the request, add info
1150          if (!request.toLowerCase().includes(chartType.toLowerCase())) {
1151              loadingEl.innerHTML = `<span class="text-sm text-blue-600">Generating a ${chartType} chart as requested...</span>
1152          }
1153      }
1154
1155      // Call backend API for Gemini visualization generation
1156      fetch('/visualization_api/', {
1157          method: 'POST',
1158          headers: {
1159              'Content-Type': 'application/json',
1160              'X-CSRFToken': getCsrfToken()
1161          },
1162          body: JSON.stringify(requestData)
1163      })
1164      .then(response => response.json())
1165      .then(data => {
```

```
        // Hide loading state
        loadingEl.classList.add('hidden');

        // Store visualization data for feedback
        window.lastVisualizationData = {
            chart_type: chartType || 'auto',
            request: request,
            success: data.success,
            has_compatibility_warning: data.has_compatibility_warning || false,
            timestamp: new Date().toISOString()
        };

        if (data.success) {
            // If successful, render the chart
            try {
                console.log("Chart config received:", data.chart_config);

                // Safely evaluate the chart configuration
                let chartConfig;
                try {
                    // First attempt: directly evaluate the config after cleaning it
                    const cleanConfig = data.chart_config
                        .replace(/Object\.assign/g, '')  // Remove Object.assign calls
                        .replace(/['"]\s*\+\s*['"]|['"]\s*\+|['"]\+\s*['"]|\+\s*['"]/g, '') // Fix string concatenation
                        .replace(/([{,])?([a-zA-Z0-9_]+)(['"])?:/g, '"$2":'); // Normalize property names to double quotes

                    chartConfig = eval('(' + cleanConfig + ')');
                } catch (evalError) {
                    console.error("Error in first evaluation attempt:", evalError);
                    try {
                        // Replace problematic quotes and syntax
                        let sanitizedConfig = data.chart_config
                            .replace(/Object\.assign/g, '')  // Remove Object.assign calls
                            .replace(/(['"])?([a-zA-Z0-9_]+)(['"])?:/g, '"$2":') // Ensure property names are quoted
                            .replace(/:\s*(['"])([^'"]*)\1/g, ':"$2"') // Ensure property values use double quotes
                            .replace(/,(\s*[}\]])/g, '$1'); // Remove trailing commas

                        chartConfig = JSON.parse(sanitizedConfig);
                    } catch (jsonError) {
                        console.error("Error in second evaluation attempt:", jsonError);

                        // Third attempt: try to fix common issues with template literals and Object references
                        try {
                            let fixedConfig = data.chart_config
                                .replace(/Object/g, '') // Remove 'Object'
                                .replace(/\.assign/g, '') // Remove '.assign'
                                .replace(/`([^`]*)`/g, '"$1"') // Replace template literals with double quotes
                                .replace(/\${([^}]*)}/g, '') // Remove template expressions
                                .replace(/\\/g, ''); // Remove escaped characters

                            if (!fixedConfig.startsWith('{')) {
                                fixedConfig = '{' + fixedConfig;
                            }
                            if (!fixedConfig.endsWith('}')) {
                                fixedConfig = fixedConfig + '}';
                            }
```

```
            // Only try to render if we have a valid chart config
            if (chartConfig && typeof chartConfig === 'object') {
                // STRICTLY enforce the selected chart type
                if (chartType) {
                    console.log(`Enforcing chart type: ${chartType} (was: ${chartConfig.type})`);
                    // Force the selected chart type to ensure it matches what the user selected
                    chartConfig.type = chartType;
                }

                // Add compatibility warning if the chart configuration suggests a different type
                if (chartType && data.has_compatibility_warning) {
                    console.log("Chart has compatibility warning");
                    // Check if the warning is already in the insights to avoid duplication
                    if (data.insights && !data.insights.includes(data.compatibility_warning)) {
                        insightsEl.innerHTML = `
                            <div class="bg-yellow-100 border-l-4 border-yellow-500 text-yellow-700 p-4 mb-4">
                                <p class="font-bold">Note about ${chartType} chart:</p>
                                <p>${data.compatibility_warning || `A ${chartType} chart may not be the optimal choice
                            </div>
                        ` + (data.insights || "");
                    } else {
                        insightsEl.innerHTML = data.insights || "";
                    }
                } else {
                    insightsEl.innerHTML = data.insights || "";
                }

```

G.3 Changing Chart Type

```
function changeChartType(chartType) {      You, last month • so progress with charts but still not folly wor…
    // Don't do anything if we haven't received the original recommendation yet
    if (!originalRecommendation) {
        console.warn("Cannot change chart type before receiving original recommendation");
        return;
    }

    const loadingEl = document.getElementById('visualizationLoading');
    const recommendationEl = document.getElementById('chartRecommendation');
    const reasonEl = document.getElementById('recommendationReason');
    const chartTypeEl = document.getElementById('recommendedChartType');

    // Show loading state
    loadingEl.classList.remove('hidden');

    // Get the current visualization request if any
    currentRequest = document.getElementById('visualizationRequest').value || '';

    // Prepare data for API call
    const requestData = {
        action: 'change_chart_type',
        dataset_info: window.datasetInfo, // Global variable set during page load
        chart_type: chartType,
        original_recommendation: JSON.stringify(originalRecommendation),
        current_request: currentRequest
    };

    console.log("Sending chart type change request:", requestData);
    // Call backend API
    fetch('/visualization_api/', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'X-CSRFToken': getCsrfToken()
        },
        body: JSON.stringify(requestData)
    })
    .then(response => response.json())
    .then(data => {
        // Hide loading state
        loadingEl.classList.add('hidden');
        console.log("Chart type change response:", data);

        if (data.success) {
            // Use our new chart type info update function to handle all UI updates
            updateChartTypeInfo(data);

            // If there is a current visualization, regenerate it with the new chart type
            if (currentRequest && currentRequest.trim() !== '') {
                // Clear previous visualization
                if (window.currentChart) {
                    window.currentChart.destroy();
                    window.currentChart = null;
                }

                // Hide visualization container
```

```
                document.getElementById('visualizationContainer').classList.add('hidden');

                // Regenerate the visualization with the new chart type
                generateVisualization(window.datasetInfo, currentRequest, chartType);
            } else {
                // If there's no current request, just clear any existing visualization
                if (window.currentChart) {
                    window.currentChart.destroy();
                    window.currentChart = null;
                }
                document.getElementById('visualizationContainer').classList.add('hidden');
                document.getElementById('visualizationError').classList.add('hidden');
            }
        } else {
            showError(data.error || "Failed to change chart type");
        }
    })
    .catch(error => {
        loadingEl.classList.add('hidden');
        showError("Network error: " + error.message);
        console.error("Error changing chart type:", error);
    });
}
```

H: Interactive Visualization Controls

H.1 Back-end connection

```
@csrf_exempt
def visualization_api(request):      You, last month • For the selected dataset Gemini gives recommend…
    """
    API endpoint for visualization recommendations and generation
    """
    if request.method != 'POST':
        return JsonResponse({'success': False, 'error': 'Only POST method is allowed'}, status=405)

    try:
        data = json.loads(request.body)
        action = data.get('action')
        dataset_info = data.get('dataset_info')

        if not action or not dataset_info:
            return JsonResponse({'success': False, 'error': 'Missing required parameters'}, status=400)

        client = get_llm_client()

        if action == 'get_recommendation':
            return get_chart_recommendation(client, dataset_info)
        elif action == 'change_chart_type':
            chart_type = data.get('chart_type')
            original_recommendation = data.get('original_recommendation')
            current_request = data.get('current_request', '')
            if not chart_type:
                return JsonResponse({'success': False, 'error': 'Missing chart type'}, status=400)
            return change_chart_type(client, dataset_info, chart_type, original_recommendation, current_request)
        elif action == 'generate_visualization':
            user_request = data.get('user_request')
            chart_type = data.get('chart_type', None)  # Get the requested chart type if provided
            if not user_request:
                return JsonResponse({'success': False, 'error': 'Missing user request'}, status=400)
            return generate_visualization(client, dataset_info, user_request, chart_type)
        else:
            return JsonResponse({'success': False, 'error': f'Unknown action: {action}'}, status=400)

    except json.JSONDecodeError:
        return JsonResponse({'success': False, 'error': 'Invalid JSON'}, status=400)
    except Exception as e:
        logging.exception("Error in visualization API: %s", str(e))
        return JsonResponse({'success': False, 'error': f'Server error: {str(e)}'}, status=500)
```

H.2 Chart Type Explanation

```python
def get_chart_type_explanation(chart_type, dataset_characteristics):
    """
    Get a specific explanation for the requested chart type based on dataset characteristics
    """
    if chart_type == 'line':
        if not dataset_characteristics.get("has_multiple_years", False):
            return "Line charts are most effective for showing trends over time. Since this dataset only contains data for a single year, you should focus on comparing values across different categories or institutions rather than showing trends over time."
        else:
            return "Line charts work best for showing trends over time or continuous data. Focus on how values change across the available years for one or more institutions."

    elif chart_type == 'bar':
        if dataset_characteristics.get("has_multiple_institutions", False) and dataset_characteristics.get("has_multiple_years", False):
            return "Bar charts are excellent for comparing values across categories. You can compare institutions side by side, or show change over time by grouping bars by year."
        else:
            return "Bar charts are ideal for comparing values across categories. Focus on comparing different metrics or institutions within the available data."

    elif chart_type == 'pie':
        if dataset_characteristics.get("has_multiple_institutions", False) and len(dataset_characteristics.get("available_institutions", [])) > 5:
            return "Pie charts work best with a small number of categories (5 or fewer) and when showing parts of a whole. Consider focusing on just a few key institutions or categories to maintain clarity."
        else:
            return "Pie charts are best for showing proportions or percentages of a whole. Focus on the distribution of a single metric across categories for a specific time period rather than comparisons over time."

    return None      You, last month • so progress with charts but still not folly wor…

# Configure logging to prevent duplicate handlers
logger = logging.getLogger('core.views')
# Remove all handlers to avoid duplicates
if logger.handlers:
    for handler in logger.handlers:
        logger.removeHandler(handler)
# Add a single handler
handler = logging.StreamHandler()
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
handler.setFormatter(formatter)
logger.addHandler(handler)
logger.propagate = False  # Prevent propagation to root logger

```

I: Dashboard Interface

I.1 Display Query Results

```
function displayQueryAnalysis(data) {
    console.log('Displaying query analysis:', data);

    // Store the data for later use
    window.currentQueryData = data;

    if (!aiQueryResults) {
        console.error('Query results container not found');
        return;
    }

    // Check if there was an API error
    if (data.api_error) {
        aiQueryResults.innerHTML = `
            <div class="bg-red-100 border-l-4 border-red-500 text-red-700 p-4 mt-4" role="alert">
                <p class="font-bold">Gemini API Error</p>
                <p>${data.api_error}</p>
                <p class="mt-2 text-sm">Using fallback analysis instead. For accurate entity extraction, please ensure
                the Gemini API key is configured correctly.</p>
            </div>
        `;

        // We'll still show the fallback results below the error message      You, 2 months ago • created indeing file fo
    }

    // Create HTML for the query analysis with collapsible section
    let resultsHTML = `
        <div class="bg-white shadow-md rounded-lg p-6 mt-8">
            <div class="flex justify-between items-center mb-4">
                <h3 class="text-lg font-semibold">
                    Query Analysis ${data.using_mock ? '(Mock AI)' : 'by Gemini AI'}
                    ${data.using_mock ?
                        '<span class="text-xs font-normal bg-yellow-100 text-yellow-800 px-2 py-1 rounded ml-2">Using
                        offline AI simulation</span>' :
                        '<span class="text-xs font-normal bg-green-100 text-green-800 px-2 py-1 rounded ml-2">Using
                        Google Gemini API</span>'}
                    ${data.cached === true ?
                        '<span class="text-xs font-normal bg-purple-100 text-purple-800 px-2 py-1 rounded ml-2">
                        Retrieved from cache</span>' : ''}
                </h3>
                <button id="queryDetailsToggle" class="bg-blue-100 hover:bg-blue-200 text-blue-800 px-3 py-1 rounded
                text-sm transition-colors duration-150 ease-in-out">
                    <span class="show-text">Show Details</span>
                    <span class="hide-text hidden">Hide Details</span>
                </button>
            </div>

            <div class="mb-4">
                <p class="font-medium text-gray-700">Your query:</p>
                <p class="text-blue-800 bg-blue-50 p-2 rounded">${data.query}</p>
            </div>

            <div id="queryDetailsContent" class="hidden">
```

```
                ${data.has_institution_typos || data.has_year_typos ? `
                <div class="mb-4 bg-blue-50 p-3 rounded border border-blue-200">
                    <p class="font-medium text-gray-700">Corrected query:</p>
                    <p class="text-blue-800 p-2">${generateCorrectedQuery(data)}</p>
                </div>
                ` : ''}

                <div class="grid grid-cols-1 md:grid-cols-3 gap-4">
                    <div class="border rounded p-3">
                        <p class="font-medium text-gray-700">Institutions:</p>
                        <ul class="list-disc list-inside">
                            ${data.institutions && data.institutions.length > 0 ?
                                data.institutions.map(inst => `<li>${inst}</li>`).join('') :
                                '<li class="text-gray-500">None specified</li>'}
                        </ul>
                    </div>

                    <div class="border rounded p-3">
                        <p class="font-medium text-gray-700">Years:</p>
                        <ul class="list-disc list-inside">
                            ${data.years && data.years.length > 0 ?
                                data.years.map(year => `<li>${year}</li>`).join('') :
                                '<li class="text-gray-500">All available years</li>'}
                        </ul>
                    </div>

                    <div class="border rounded p-3">
                        <p class="font-medium text-gray-700">Year Range:</p>
                        <p class="${data.start_year ? 'font-semibold text-blue-700' : 'text-gray-500'}">
                            ${yearRangeDisplay}
                        </p>
                    </div>
                </div>

                ${data.mission_group ? `
                <div class="mt-4 border rounded p-3 bg-blue-50">
                    <p class="font-medium text-gray-700">Mission Group Filter:</p>
                    <p class="font-semibold text-blue-700">${data.mission_group}</p>
                    <p class="text-sm mt-1">Including data for ${data.mission_group_institutions?.length || 0}
                    institutions from this group</p>
                </div>
                ` : ''}

                <div class="mt-4 border rounded p-3">
                    <p class="font-medium text-gray-700">Data requested:</p>
                    <p class="font-semibold ${dataRequestDisplay === 'General information' ? 'text-gray-500' :
                    'text-blue-700'}">
                        ${dataRequestDisplay}
                    </p>
                </div>

                <div class="mt-4 text-sm text-gray-600 border-t pt-2">
                    <p>Entity extraction performed by ${data.using_mock ?
                        'local AI simulation (Gemini API unavailable)' :
                        'Google\'s Gemini AI'}</p>
                    ${data.cached === true ?
                        '<p>Results retrieved from cache. Last processed: ' + new Date().toLocaleString() + '</p>' : ''}
                </div>
```

```
        // Add feedback component at the top
        resultsHTML = `
            <div id="feedbackContainer" class="bg-yellow-50 border border-yellow-200 rounded-lg p-6 mb-6 mt-6">
                <h4 class="text-lg font-semibold mb-2">How helpful were these results?</h4>
                <div class="flex flex-wrap gap-2">
                    <button id="veryHelpfulBtn" class="feedback-btn bg-green-100 hover:bg-green-200 text-green-800 px-4
                    rounded-md" data-value="Very Helpful">Very Helpful</button>
                    <button id="helpfulBtn" class="feedback-btn bg-blue-100 hover:bg-blue-200 text-blue-800 px-4 py-2
                    rounded-md" data-value="Helpful">Helpful</button>
                    <button id="notHelpfulBtn" class="feedback-btn bg-red-100 hover:bg-red-200 text-red-800 px-4 py-2
                    rounded-md" data-value="Not Helpful">Not Helpful</button>
                </div>
            </div>
        ` + resultsHTML;
```

I.2 Reviewable Dataset

```
Windsurf: Refactor | Explain | ×
720     function renderFilePreview(dataset, datasetIndex) {       You, last month • important stage …
721         console.log(`Rendering file previews for dataset ${datasetIndex + 1}: ${dataset.title || 'Untitled'}`);
722
723         const filesContainer = document.getElementById(`files-container-${datasetIndex}`);
724         if (!filesContainer) {
725             console.error(`Files container not found for dataset ${datasetIndex}`);
726             return;
727         }
728
729         if (!dataset.matches || dataset.matches.length === 0) {
730             filesContainer.innerHTML = `
731                 <div class="text-sm text-gray-500 p-3 bg-gray-50 rounded">
732                     No files available
733                 </div>
734             `;
735             return;
736         }
737
738         let filesHTML = '';
739
740         dataset.matches.forEach(match => {
741             console.log(`Processing match for dataset ${datasetIndex + 1}:`, match);
742
743             filesHTML += `
744                 <div class="mt-4 mb-4 border rounded p-3 bg-white">
745                     <div class="text-sm font-medium mb-2">${match.academic_year || 'Unknown Year'} - ${match.reference ||
                        'Unknown Reference'}</div>
747
748                 if (match.preview && match.preview.columns && match.preview.data) {
749                     console.log(`Processing preview data for ${match.reference}`, {
750                         columns: match.preview.columns.length,
751                         data: match.preview.data.length
752                     });
753
754                     try {
755                         // Clean metadata artifacts from columns and data
756                         const { columns: uniqueColumns, data: previewData } = cleanMetadataArtifacts(
757                             match.preview.columns,
758                             match.preview.data
759                         );
760
761                         filesHTML += `
762                             <div class="overflow-x-auto max-h-[300px] table-container border rounded">
763                                 <table class="min-w-full border-collapse table-auto text-sm">
764                                     <thead>
765                                         <tr>
766                                             ${uniqueColumns.map(column =>
767                                                 `<th class="px-4 py-2 border-b border-gray-300 text-left text-sm font-medium
                                                    sticky top-0 bg-white z-10">${column}</th>`
768                                             ).join('')}
769                                         </tr>
770                                     </thead>
771                                     <tbody>
772                                         ${previewData.length > 0 ?
773                                             previewData.map(row => `
774                                                 <tr>
```

```
                                        ${row.map(cell =>
                                            `<td class="px-4 py-2 border-b border-gray-300 text-sm">${cell}</td>`
                                        ).join('')}
                                    </tr>
                                `).join('') :
                                `<tr><td colspan="${uniqueColumns.length}" class="px-4 py-2 text-center
                                text-gray-500">No data available or no matching institutions found</td></tr>`
                            }
                        </tbody>
                    </table>
                </div>
                ${match.preview.has_more ?
                    `<div class="text-xs text-gray-500 mt-2">
                        Showing ${previewData.length} of ${match.preview.matched_rows || previewData.length}
                        matching rows
                    </div>` :
                    ''
                }
            `;
        } catch (error) {
            console.error('Error processing preview data:', error);
            filesHTML += `
                <div class="text-sm text-red-500 p-3 bg-red-50 rounded">
                    Error processing preview data: ${error.message}
                </div>
            `;
        }
    } else if (match.preview && match.preview.error) {
        filesHTML += `
            <div class="text-sm text-red-500 p-3 bg-red-50 rounded">
                Error loading preview: ${match.preview.error}
            </div>
        `;
    } else {
        filesHTML += `
            <div class="text-sm text-gray-500 p-3 bg-gray-50 rounded">
                Preview not available
            </div>
        `;
    }

    filesHTML += `</div>`;
});

filesContainer.innerHTML = filesHTML;
}
```

I.3 Manage loading states

```
function showLoading(message = 'Processing your query...') {
    console.log('Showing loading indicator');
    let loadingIndicator = document.getElementById('aiLoadingIndicator');
    if (!loadingIndicator) {
        // Create loading indicator if it doesn't exist
        loadingIndicator = document.createElement('div');
        loadingIndicator.id = 'aiLoadingIndicator';
        loadingIndicator.className = 'fixed top-0 left-0 w-full h-full flex items-center justify-center bg-gray-800 bg-op
        loadingIndicator.innerHTML = `
            <div class="backdrop-blur-md bg-gray-800 bg-opacity-20 p-8 rounded-lg shadow-lg flex flex-col items-center
                <div class="w-16 h-16 mb-4">
                    <svg class="animate-spin w-full h-full text-blue-500" xmlns="http://www.w3.org/2000/svg" fill="none
                        <circle class="opacity-25" cx="12" cy="12" r="10" stroke="currentColor" stroke-width="4"></circle
                        <path class="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0 12h4zm2 5.2
                    </svg>
                </div>
                <h2 class="text-xl font-semibold mb-2 text-white">Analyzing Data</h2>
                <p class="text-gray-200 text-center">${message}</p>
            </div>
        `;
        document.body.appendChild(loadingIndicator);
        console.log('Loading indicator created and added to page');
    } else {
        const messageElement = loadingIndicator.querySelector('p');
        if (messageElement) {
            messageElement.textContent = message;
        }
        // Show existing loading indicator
        loadingIndicator.classList.remove('hidden');
```

I.4 Query Input

```
88      if (aiSearchBtn && aiQueryInput) {
89          aiSearchBtn.addEventListener('click', function() {      You, 2 months ago • created indeing file for the AI approch.
90              const query = aiQueryInput.value.trim();
91
92              console.log('AI Search button clicked with query:', query);
93
94              if (!query) {
95                  alert('Please enter a query');
96                  return;
97              }
98
99              // Record the start time of the query processing
100             window.queryStartTime = performance.now();
101             console.log('Query processing started at:', window.queryStartTime);
102
103             // Show loading state
104             showLoading('Analyzing your query with Gemini AI...');
105
106             // Prepare max matches parameter
107             const maxMatches = aiMaxMatches ? aiMaxMatches.value : 3;
108
109             // Get selected mission group
110             const selectedMissionGroup = document.querySelector('input[name="missionGroup"]:checked').value;
111             console.log('Selected mission group:', selectedMissionGroup);
112
113             // Call the Gemini API endpoint
114             fetch('/process_gemini_query/', {
115                 method: 'POST',
116                 headers: {
117                     'Content-Type': 'application/json',
118                     'X-CSRFToken': getCsrfToken(),
119                 },
120                 body: JSON.stringify({
121                     query: query,
122                     max_matches: maxMatches,
123                     mission_group: selectedMissionGroup === 'none' ? null : selectedMissionGroup,
124                     start_time: window.queryStartTime
125                 })
126             })
127             .then(response => {
128                 console.log('Received response with status:', response.status);
129
130                 if (!response.ok) {
131                     return response.text().then(text => {
132                         console.error('Error response body:', text);
133                         throw new Error('Network response was not ok: ' + response.status);
134                     });
135                 }
136                 return response.json();
137             })
138             .then(data => {
139                 console.log('Received data from API:', data);
140                 // Calculate the end time and total duration
141                 const endTime = performance.now();
142                 const duration = endTime - window.queryStartTime;
143                 console.log(`Query completed in ${duration.toFixed(2)} ms`);
144
145                 // Send timing information to the server
```

```
146                 recordQueryTiming(query, duration, data.matching_datasets ? data.matching_datasets.length : 0, selectedMissio
147
148                 hideLoading();
149
150                 if (data.status === 'error') {
151                     console.error('Error from backend:', data.error);
152                     showError(data.error);
153                     return;
154                 }
155
156                 // Display the query analysis
157                 displayQueryAnalysis(data);
158             })
159             .catch(error => {
160                 console.error('Fetch error:', error);
161                 hideLoading();
162                 showError('An error occurred while processing your query. Please try again.');
163             });
164         });
165     }
```

I.5: Exporting data

```
618 document.addEventListener('DOMContentLoaded', function() {      You, last month • making it possible to download data from se
619     // CSV download buttons
620     document.querySelectorAll('.download-csv-btn').forEach(button => {
621         button.addEventListener('click', function(e) {
622             // Prevent event from propagating to document level handlers
623             e.stopPropagation();
624
625             const tableId = this.getAttribute('data-table-id');
626             const tableIndex = tableId.split('-').pop();
627             const filename = `dataset_${tableIndex}.csv`;
628             downloadTableAsCSV(tableId, filename);
629
630             // Prevent default button behavior
631             e.preventDefault();
632         });
633     });
634
635     // PDF download buttons
636     document.querySelectorAll('.download-pdf-btn').forEach(button => {
637         button.addEventListener('click', function(e) {
638             // Prevent event from propagating to document level handlers
639             e.stopPropagation();
640
641             const tableId = this.getAttribute('data-table-id');
642             const tableIndex = tableId.split('-').pop();
643             const filename = `dataset_${tableIndex}.pdf`;
644             downloadTableAsPDF(tableId, filename);
645
646             // Prevent default button behavior
647             e.preventDefault();
648         });
649     });
650
651     // Excel download buttons
652     document.querySelectorAll('.download-excel-btn').forEach(button => {
653         button.addEventListener('click', function(e) {
654             // Prevent event from propagating to document level handlers
655             e.stopPropagation();
656
657             const tableId = this.getAttribute('data-table-id');
658             const tableIndex = tableId.split('-').pop();
659             const filename = `dataset_${tableIndex}.xlsx`;
660             downloadTableAsExcel(tableId, filename);
661
662             // Prevent default button behavior
663             e.preventDefault();
664         });
665     });
666 });
```

I.6 Prepare Data for Clean Display

```
function cleanMetadataArtifacts(columns, tableData) {      You, last month • important stage …
    console.log('Cleaning metadata artifacts from columns:', columns);

    // Handle case where no columns are provided
    if (!columns || columns.length === 0) {
        console.warn('No columns provided to cleanMetadataArtifacts');
        return { columns: [], data: tableData || [] };
    }

    // Remove any metadata string from column headers
    const cleanColumns = columns.map(col => {
        // Remove #METADATA and any JSON that follows it
        if (typeof col === 'string') {
            // Handle metadata in column headers
            Test Regex...
            const metadataRemoved = col.replace(/#METADATA:.*?}/g, '').trim();

            // Make column names look nicer (capitalize first letter of each word)
            Test Regex...
            return metadataRemoved.split(/[\s_]+/).map(word =>
                word.charAt(0).toUpperCase() + word.slice(1).toLowerCase()
            ).join(' ');
        }
        return col;
    });

    console.log('Columns after metadata removal:', cleanColumns);
    // Remove duplicate "Academic Year" columns
    const uniqueColumns = [];
    const seenColumns = new Set();

    cleanColumns.forEach(col => {
        if (!seenColumns.has(col)) {
            uniqueColumns.push(col);
            seenColumns.add(col);
        } else {
            console.log(`Duplicate column removed: ${col}`);
        }
    });

    console.log('Final unique columns:', uniqueColumns);

    // Clean row data - remove any metadata strings and ensure correct length
    const cleanData = (tableData || []).map(row => {
        // Handle metadata in row data
        const cleanRow = row.map(cell => {
            if (typeof cell === 'string') {
                Test Regex...
                return cell.replace(/#METADATA:.*?}/g, '').trim();
            }
            return cell;
        });

        // Adjust row length to match columns
        if (cleanRow.length > uniqueColumns.length) {
            console.log(`Trimming row from ${cleanRow.length} to ${uniqueColumns.length} columns`);
            return cleanRow.slice(0, uniqueColumns.length);
        } else if (cleanRow.length < uniqueColumns.length) {
            console.log(`Padding row from ${cleanRow.length} to ${uniqueColumns.length} columns`);
            const paddedRow = [...cleanRow];
            while (paddedRow.length < uniqueColumns.length) {
                paddedRow.push('');
            }
            return paddedRow;
        }
        return cleanRow;
    });

    return { columns: uniqueColumns, data: cleanData };
}
```

I.7 Dynamic dataset preview toggle

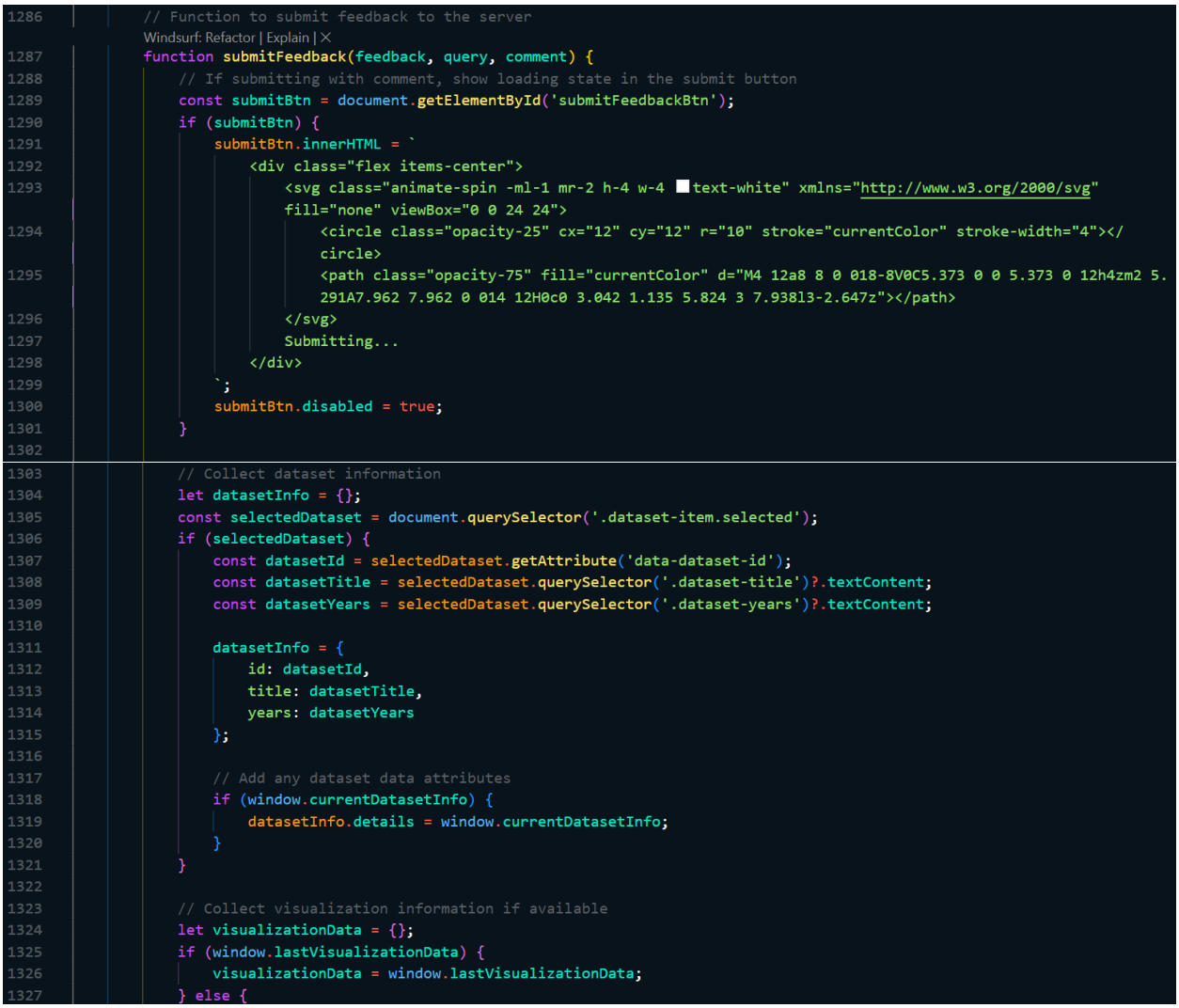
```
toggleBtn.addEventListener('click', function() {
    // Toggle content visibility
    contentDiv.classList.toggle('hidden');

    // Toggle button text
    showText.classList.toggle('hidden');
    hideText.classList.toggle('hidden');

    // Add smooth animation (if CSS is defined for it)
    if (!contentDiv.classList.contains('hidden')) {
        contentDiv.classList.add('animate-fade-in');
        setTimeout(() => {
            contentDiv.classList.remove('animate-fade-in');
        }, 300);
    }
});
```

J: Feedback System

```
1229    function setupFeedbackButtons(query) {      You, 4 weeks ago • working on feedback logging …
1230        const feedbackButtons = document.querySelectorAll('.feedback-btn');
1231
1232        feedbackButtons.forEach(button => {
1233            button.addEventListener('click', function() {
1234                const feedback = this.textContent.trim();
1235
1236                // Disable all buttons to prevent multiple selections
1237                feedbackButtons.forEach(btn => btn.disabled = true);
1238
1239                // If "Helpful" or "Not Helpful" is clicked, show comment textarea
1240                if (feedback === "Helpful" || feedback === "Not Helpful") {
1241                    // Create comment section
1242                    const commentSection = document.createElement('div');
1243                    commentSection.className = 'mt-4';
1244                    commentSection.innerHTML = `
1245                        <p class="mb-2 font-medium text-gray-700">How could we improve these results?</p>
1246                        <textarea id="feedbackComment" class="w-full px-3 py-2 border border-gray-300 rounded-md
                        focus:outline-none focus:ring-2 focus:ring-blue-500"
1247                            rows="3" placeholder="Your comments will help us improve our search results (optional)"></
                            textarea>
1248                        <div class="mt-2 flex justify-end">
1249                            <button id="submitFeedbackBtn" class="bg-blue-600 hover:bg-blue-700 text-white px-4 py-2
                            rounded-md">
1250                                Submit Feedback
1251                            </button>
1252                        </div>
1253                    `;
1255                    // Add to container
1256                    document.getElementById('feedbackContainer').appendChild(commentSection);
1257
1258                    // Enable the selected button with visual indication
1259                    this.disabled = false;
1260                    this.classList.add('ring-2', 'ring-blue-500');
1261
1262                    // Handle submit button click
1263                    document.getElementById('submitFeedbackBtn').addEventListener('click', function() {
1264                        const commentText = document.getElementById('feedbackComment').value;
1265                        submitFeedback(feedback, query, commentText);
1266                    });
1267                } else {
1268                    // If "Very Helpful" is clicked, submit immediately without comment
1269                    // Show loading state in the button
1270                    const originalText = this.textContent;
1271                    this.innerHTML = `
1272                        <div class="flex items-center">
1273                            <svg class="animate-spin -ml-1 mr-2 h-4 w-4 text-current" xmlns="http://www.w3.org/2000/svg"
                            fill="none" viewBox="0 0 24 24">
1274                                <circle class="opacity-25" cx="12" cy="12" r="10" stroke="currentColor" stroke-width="4"></
                                circle>
1275                                <path class="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0
                                12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path>
1276                            </svg>
1277                            Submitting...
1278                        </div>
1279                    `;
```



```
1286    // Function to submit feedback to the server
        Windsurf: Refactor | Explain | ×
1287    function submitFeedback(feedback, query, comment) {
1288        // If submitting with comment, show loading state in the submit button
1289        const submitBtn = document.getElementById('submitFeedbackBtn');
1290        if (submitBtn) {
1291            submitBtn.innerHTML = `
1292                <div class="flex items-center">
1293                    <svg class="animate-spin -ml-1 mr-2 h-4 w-4 text-white" xmlns="http://www.w3.org/2000/svg"
                        fill="none" viewBox="0 0 24 24">
1294                        <circle class="opacity-25" cx="12" cy="12" r="10" stroke="currentColor" stroke-width="4"></
                        circle>
1295                        <path class="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0 12h4zm2 5.
                        291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path>
1296                    </svg>
1297                    Submitting...
1298                </div>
1299            `;
1300            submitBtn.disabled = true;
1301        }
1302
1303        // Collect dataset information
1304        let datasetInfo = {};
1305        const selectedDataset = document.querySelector('.dataset-item.selected');
1306        if (selectedDataset) {
1307            const datasetId = selectedDataset.getAttribute('data-dataset-id');
1308            const datasetTitle = selectedDataset.querySelector('.dataset-title')?.textContent;
1309            const datasetYears = selectedDataset.querySelector('.dataset-years')?.textContent;
1310
1311            datasetInfo = {
1312                id: datasetId,
1313                title: datasetTitle,
1314                years: datasetYears
1315            };
1316
1317            // Add any dataset data attributes
1318            if (window.currentDatasetInfo) {
1319                datasetInfo.details = window.currentDatasetInfo;
1320            }
1321        }
1322
1323        // Collect visualization information if available
1324        let visualizationData = {};
1325        if (window.lastVisualizationData) {
1326            visualizationData = window.lastVisualizationData;
1327        } else {
```

```
        // Send feedback to the backend
        fetch('/save_feedback/', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'X-CSRFToken': getCsrfToken(),
            },
            body: JSON.stringify({
                query: query,
                feedback: feedback,
                comment: comment,
                dataset_info: datasetInfo,
                visualization_data: visualizationData,
                window_query_data: window.currentQueryData || {}
            })
        })
        .then(response => {
            if (!response.ok) {
                throw new Error(`Network response was not ok: ${response.status}`);
            }
            return response.json();
        })
        .then(data => {
            // Replace the entire feedback container with a thank you message
            const feedbackContainer = document.getElementById('feedbackContainer');
            if (feedbackContainer) {
                feedbackContainer.innerHTML = `
                    <div class="text-green-700">
                        <p class="font-medium">Thanks for your feedback!</p>
                        <p class="text-sm">Your input helps us improve our search results.</p>
                    </div>
                `;
            }

            // Store visualization data for feedback
            const chartTypeButtons = document.querySelectorAll('.chart-type-button');
            const currentChartType = Array.from(chartTypeButtons).find(btn => btn.classList.contains('selected'))?.
            getAttribute('data-chart-type');

            window.lastVisualizationData = {
                chart_type: visualizationData?.chart_type || currentChartType,
                request: document.getElementById('visualizationRequest')?.value || '',
                success: data.success,
                has_compatibility_warning: data.has_compatibility_warning || false
            };
        })
        .catch(error => {
            console.error('Error submitting feedback:', error);
            // Show error message
            const feedbackContainer = document.getElementById('feedbackContainer');
            if (feedbackContainer) {
                feedbackContainer.innerHTML += `
                    <div class="text-red-600 mt-2">
                        <p>Failed to submit feedback. Please try again.</p>
                    </div>
                `;
            }
```

K: Caching

```python
class QueryCache:      You, last month • implemented cahcing for queries, cahed being st…
    """
    A simple file-based cache for query results with expiration handling.
    Caches query results in JSON format in the data/cache directory.
    """

    Windsurf: Refactor | Explain | ×
    def __init__(self):
        """Initialize the query cache with the cache directory path."""
        # Get base directory from settings or use default
        base_dir = getattr(settings, 'BASE_DIR', None)
        if base_dir:
            self.cache_dir = Path(base_dir) / 'data' / 'cache'
        else:
            # Fallback to a relative path if BASE_DIR is not defined
            self.cache_dir = Path(__file__).resolve().parent.parent.parent.parent / 'data' / 'cache'

        # Ensure the cache directory exists
        os.makedirs(self.cache_dir, exist_ok=True)
        logger.info(f"Query cache initialized with directory: {self.cache_dir}")

    def _generate_cache_key(self, query):
        """
        Generate a unique cache key for a query.

        """
        # Create an MD5 hash of the query string to use as the filename
        # This ensures unique filenames and handles special characters
        hash_obj = hashlib.md5(query.encode('utf-8'))
        return hash_obj.hexdigest()

    Windsurf: Refactor | Explain | ×
    def _get_cache_file_path(self, cache_key):
        """
        Get the full path to a cache file.

        """
        return self.cache_dir / f"{cache_key}.json"

    def get(self, query):
        """
        Retrieve a cached query result if it exists and hasn't expired.

        """
        cache_key = self._generate_cache_key(query)
        cache_file = self._get_cache_file_path(cache_key)

        if not cache_file.exists():
            logger.debug(f"Cache miss for query: {query[:50]}...")
            return None

        try:
            with open(cache_file, 'r', encoding='utf-8') as f:
                cached_data = json.load(f)

            # Check if the cache has expired (30 days)
            cached_time = datetime.datetime.fromisoformat(cached_data.get('cached_at', ''))
            expiration = datetime.timedelta(days=30)

            if datetime.datetime.now() - cached_time > expiration:
                logger.debug(f"Cache expired for query: {query[:50]}...")
                # Remove the expired cache file
                os.remove(cache_file)
                return None

            logger.debug(f"Cache hit for query: {query[:50]}...")
```

```
            # Update access count and last accessed time
            cached_data['access_count'] = cached_data.get('access_count', 0) + 1
            cached_data['last_accessed'] = datetime.datetime.now().isoformat()

            # Write the updated metadata back to the cache
            with open(cache_file, 'w', encoding='utf-8') as f:
                json.dump(cached_data, f, ensure_ascii=False, indent=2)

            return cached_data.get('result')

        except (json.JSONDecodeError, KeyError, ValueError) as e:
            logger.error(f"Error reading cache file {cache_file}: {str(e)}")
            # Remove invalid cache file
            if cache_file.exists():
                os.remove(cache_file)
            return None

    Windsurf: Refactor | Explain | ×
    def set(self, query, result):
        """
        Cache a query result.


        """
        cache_key = self._generate_cache_key(query)
        cache_file = self._get_cache_file_path(cache_key)

        try:
            # Create cache data structure with metadata
            cache_data = {
                'query': query,
                'result': result,
                'cached_at': datetime.datetime.now().isoformat(),
                'access_count': 0,
                'last_accessed': None
            }

            # Write to cache file
            with open(cache_file, 'w', encoding='utf-8') as f:
                json.dump(cache_data, f, ensure_ascii=False, indent=2)

            logger.debug(f"Cached result for query: {query[:50]}...")
            return True

        except Exception as e:
            logger.error(f"Error caching query result: {str(e)}")
            return False

    Windsurf: Refactor | Explain | ×
    def clear(self, query=None):
        """
        Clear specific cache entry or all cache entries.

        """
        if query:
            # Clear specific cache entry
            cache_key = self._generate_cache_key(query)
```

```
            cache_file = self._get_cache_file_path(cache_key)

            if cache_file.exists():
                os.remove(cache_file)
                logger.info(f"Cleared cache for query: {query[:50]}...")
                return 1
            return 0
        else:
            # Clear all cache entries
            count = 0
            for cache_file in self.cache_dir.glob('*.json'):
                os.remove(cache_file)
                count += 1

            logger.info(f"Cleared {count} entries from the query cache")
            return count

    Windsurf: Refactor | Explain | ×
    def get_cache_stats(self):
        """
        Get statistics about the current cache.

        """
        stats = {
            'total_entries': 0,
            'total_size_bytes': 0,
            'oldest_entry': None,
            'newest_entry': None,
            'most_accessed': None,
            'most_accessed_count': 0
            'most_accessed_count': 0
        }

        # Get all cache files
        cache_files = list(self.cache_dir.glob('*.json'))
        stats['total_entries'] = len(cache_files)

        oldest_time = None
        newest_time = None

        for cache_file in cache_files:
            # Get file size
            file_size = os.path.getsize(cache_file)
            stats['total_size_bytes'] += file_size

            try:
                with open(cache_file, 'r', encoding='utf-8') as f:
                    cache_data = json.load(f)

                cached_time = datetime.datetime.fromisoformat(cache_data.get('cached_at', ''))

                # Track oldest and newest entries
                if oldest_time is None or cached_time < oldest_time:
                    oldest_time = cached_time
                    stats['oldest_entry'] = {
                        'query': cache_data.get('query', '')[:50] + '...',
                        'cached_at': cached_time.isoformat()
                    }

                if newest_time is None or cached_time > newest_time:
```

```
                    newest_time = cached_time
                    stats['newest_entry'] = {
                        'query': cache_data.get('query', '')[:50] + '...',
                        'cached_at': cached_time.isoformat()
                    }

                # Track most accessed entry
                access_count = cache_data.get('access_count', 0)
                if access_count > stats['most_accessed_count']:
                    stats['most_accessed_count'] = access_count
                    stats['most_accessed'] = {
                        'query': cache_data.get('query', '')[:50] + '...',
                        'access_count': access_count
                    }

            except (json.JSONDecodeError, KeyError) as e:
                logger.warning(f"Error reading cache file {cache_file}: {str(e)}")

        # Convert total size to a more readable format
        if stats['total_size_bytes'] > 1024 * 1024:
            stats['total_size'] = f"{stats['total_size_bytes'] / (1024 * 1024):.2f} MB"
        else:
            stats['total_size'] = f"{stats['total_size_bytes'] / 1024:.2f} KB"

        return stats
```

L: Gemini Client

```python
class GeminiClient:      You, 2 months ago • created indeing file for the AI approch. fix th…
    """Client for interacting with Google's Gemini API for natural language processing."""

    Windsurf: Refactor | Explain | ×
    def __init__(self, api_key: Optional[str] = None):
        """Initialize the Gemini client with API key."""
        # Try to get API key from environment variable if not provided
        self.api_key = api_key or os.environ.get('GEMINI_API_KEY')
        if not self.api_key:
            logger.warning("No Gemini API key provided. API calls will not work.")

    Windsurf: Refactor | Explain | ×
    def analyze_query(self, query: str) -> Dict[str, Any]:
        """
        Analyze a natural language query using the Gemini API.

        """
        import requests
        import json
        import re
        import logging

        logger = logging.getLogger(__name__)
        logger.info(f"Analyzing query with Gemini API: {query}")

        if not self.api_key:
            logger.error("No Gemini API key provided")
            raise ValueError("Gemini API key is required")
        # Construct the prompt
        system_prompt = """
        You are a helpful assistant that extracts structured information from a user's query about higher education
        statistics.

        Extract the following information and output it in JSON format:
        1. Institutions mentioned (e.g., "University of Oxford", "University of Leicester")
        2. Years mentioned (e.g., "2020", "2020/21")
        3. If a year range is given, identify the start_year and end_year
        4. The type of data being requested (e.g., "student numbers", "enrollment", "graduates")

        IMPORTANT: You need to detect and correct ONLY SPELLING typos in institution names and years.

        For institutions, follow these strict rules:
        - ONLY consider a term to have a typo if it has clear misspellings (e.g., "Univercity" → "University", "Liecester" →
        "Leicester")
        - DO NOT consider "Oxford University" vs "University of Oxford" as a typo - these are different ways to refer to the
        same institution
        - DO NOT mark an institution as having a typo if it's spelled correctly but uses a different naming convention
        - DO NOT transform a city name like "london" into "The University of London"
        - PRESERVE the original terms from the query - if user says "london", keep it as "london" in the institutions list
        - If there are no spelling errors, original_institutions should exactly match institutions, and has_institution_typos
        should be false      You, last month • important stage …

        For years:
        - ONLY mark years as having typos if they have clear numerical errors (e.g., "20025" → "2025")
        - If no year typos exist, original_years should exactly match years, and has_year_typos should be false

```

```
    Be careful about interpreting years in academic context:
    - If the query contains phrases like "starting in [YEAR]" or "beginning in [YEAR]", interpret [YEAR] as the start of an academic year.
    - If the query contains phrases like "end of [YEAR]" or "ending in [YEAR]", interpret [YEAR] as the end of an academic year.
    - For ranges like "2016 to 2017", treat both as starting years of academic years.
    - If there's no clarification, assume a year refers to the starting year of an academic year.
    - For "past X years", calculate the years based on the current year (2025).

    For the data_request field, use the SPECIFIC TERMS from the query whenever possible:
    - If the query mentions "undergraduates", use ["undergraduate"] not ["student_enrollment"]
    - If the query mentions "postgraduates", use ["postgraduate"] not ["student_enrollment"]
    - If the query mentions "student numbers", use ["student_count"]
    - If the query mentions "enrollment" or "enrolment", use ["enrollment"]
    - If the query mentions "staff" or "teachers", use ["staff_data"]
    - If the query mentions "research", use ["research_data"]
    - NEVER generalize specific educational levels - keep them exactly as requested

    Output format:
    {
      "institutions": ["University X", "london"],
      "original_institutions": ["Universcity X", "london"],
      "has_institution_typos": true,
      "years": ["2019/20", "2020/21"],
      "original_years": ["20019", "2020"],
      "has_year_typos": true,
      "start_year": "2019",
      "end_year": "2020",
      "data_request": ["undergraduate", "graduation_rates"]
    }

    If no specific institutions are mentioned, return an empty list for institutions and original_institutions.
    If no typos were detected, original_institutions should match institutions, and has_institution_typos should be false.
    The same applies to years and original_years.
    """

    # The actual user query
    user_prompt = f"Extract information from this query, paying special attention to academic year conventions and potential typos in institution names and years: '{query}'"

    try:
        # Gemini API endpoint - Updated to use the most current endpoint
        url = "https://generativelanguage.googleapis.com/v1/models/gemini-1.5-pro:generateContent"

        # Request payload
        payload = {
            "contents": [
                {
                    "role": "user",
                    "parts": [
                        {"text": system_prompt},
                        {"text": user_prompt}
                    ]
                }
            ],
            "generationConfig": {
                "temperature": 0.1,
                "topP": 0.95,
```

```
                "topK": 40,
                "maxOutputTokens": 2048
            }
        }

        # Headers with API key
        headers = {
            "Content-Type": "application/json",
            "x-goog-api-key": self.api_key
        }

        # Make the request
        response = requests.post(url, json=payload, headers=headers)

        # Check for successful response
        if response.status_code != 200:
            logger.error(f"Gemini API error: {response.status_code} - {response.text}")
            raise Exception(f"Gemini API returned status code {response.status_code}: {response.text}")

        # Parse the response
        response_data = response.json()

    def _process_academic_year_logic(self, query, result):
        """
        Process academic year logic based on context clues in the query.
        This handles cases like:
        - "starting in 2017" → 2017/18
        - "end of 2017" → 2016/17
        - "2016 to 2017" → 2016/17 - 2017/18 (both as starting years)

        """
        import re
        import logging

        logger = logging.getLogger(__name__)
        query_lower = query.lower()

        # Extract years with context
        start_patterns = [
            r'start(?:ing|s|ed)?\s+(?:in|from|at)?\s+(?:the\s+)?(?:year\s+)?(\d{4})',
            r'begin(?:ning|s)?\s+(?:in|from|at)?\s+(?:the\s+)?(?:year\s+)?(\d{4})',
            r'from\s+(?:the\s+)?(?:year\s+)?(\d{4})'
        ]

        end_patterns = [
            r'end(?:ing|s|ed)?\s+(?:in|at|of)?\s+(?:the\s+)?(?:year\s+)?(\d{4})',
            r'finish(?:ing|es|ed)?\s+(?:in|at)?\s+(?:the\s+)?(?:year\s+)?(\d{4})',
            r'(?:in|at)\s+(?:the\s+)?end\s+of\s+(?:the\s+)?(?:year\s+)?(\d{4})'
        ]

        # Handle range logic for years without context
        if 'start_year' in result and 'end_year' in result and result['start_year'] is not None and result['end_year'] is not
        None:
            # Range years should be treated as starting years of academic years
            start_year = result['start_year']
            end_year = result['end_year']

            try:
                # Clear existing years array to rebuild it with correct academic years
                result['years'] = []

                # Add academic years for the range
                for year in range(int(start_year), int(end_year) + 1):
                    academic_year = f"{year}/{str(year+1)[2:4]}"
                    result['years'].append(academic_year)

                logger.info(f"Processed year range: {start_year}-{end_year} as academic years: {', '.join(result['years'])}")
            except (ValueError, TypeError) as e:
                logger.error(f"Error processing year range: {e}. start_year={start_year}, end_year={end_year}")

            return

```

```python
    def get_completion(self, prompt):
        """
        Get a completion from the Gemini API based on the provided prompt.

        """
        import requests
        import logging

        logger = logging.getLogger(__name__)
        logger.info("Sending completion request to Gemini API")

        if not self.api_key:
            logger.error("No Gemini API key provided")
            raise ValueError("Gemini API key is required")

        try:
            # Updated Gemini API endpoint
            url = "https://generativelanguage.googleapis.com/v1/models/gemini-1.5-pro:generateContent"

            # Request payload
            payload = {
                "contents": [
                    {
                        "role": "user",
                        "parts": [
                            {"text": prompt}
                        ]
                    }
                ],
                "generationConfig": {
                    "temperature": 0.2,
                    "topP": 0.95,
                    "topK": 40,
                    "maxOutputTokens": 4096
                }
            }

            # Headers with API key
            headers = {
                "Content-Type": "application/json",
                "x-goog-api-key": self.api_key
            }

            # Make the request
            response = requests.post(url, json=payload, headers=headers)

            # Check for successful response
            if response.status_code != 200:
                logger.error(f"Gemini API error: {response.status_code} - {response.text}")
                raise Exception(f"Gemini API returned status code {response.status_code}: {response.text}")

            # Parse the response
            response_data = response.json()

```

```
            # Extract text from response
            if 'candidates' in response_data and len(response_data['candidates']) > 0:
                if 'content' in response_data['candidates'][0]:
                    response_text = response_data['candidates'][0]['content']['parts'][0]['text']
                    return response_text
                else:
                    logger.error("No content in Gemini API response")
                    raise Exception("No content in Gemini API response")
            else:
                logger.error("No candidates in Gemini API response")
                raise Exception("No candidates in Gemini API response")

        except Exception as e:
            logger.error(f"Error getting completion from Gemini API: {str(e)}")
            raise

    Windsurf: Refactor | Explain | X
    def generate_text(self, prompt):
        """
        Generate text using the Gemini API - alias for get_completion for compatibility

        """
        return self.get_completion(prompt)

```

M: Timing Queries

M.1 Recording the time of query processing

```python
@csrf_exempt
@require_http_methods(["POST"])
def record_query_timing(request):      You, 6 days ago • uploading all the cleaning csv files nad final …
    """
    Record timing information for query processing.
    Data is stored in a JSON file at the root level.
    """
    import json
    import os
    import logging
    import traceback
    from pathlib import Path
    from django.conf import settings

    logger = logging.getLogger(__name__)
    logger.info("=== RECORDING QUERY TIMING ===")

    try:
        # Parse the data from the request
        data = json.loads(request.body.decode('utf-8'))
        query = data.get('query', '')
        duration_ms = data.get('duration_ms', 0)
        # Store the requested match count, not the returned match count
        requested_match_count = data.get('requested_match_count', 0)
        mission_group = data.get('mission_group')

        # Validate incoming data
        if not query:
            logger.warning("Empty query received in timing data")
            return JsonResponse({'status': 'error', 'error': 'Empty query'}, status=400)

        if not isinstance(duration_ms, (int, float)) or duration_ms <= 0:
            logger.warning(f"Invalid duration value: {duration_ms}")
            return JsonResponse({'status': 'error', 'error': 'Invalid duration'}, status=400)

        # Convert to integer milliseconds
        duration_ms_int = int(duration_ms)

        # Format duration in seconds with 2 decimal places
        duration_sec = round(duration_ms / 1000, 2)

        # Define the path to the timing data file - ensure it's in the project root
        timing_file_path = Path(settings.BASE_DIR) / 'timing_queries.json'

        logger.info(f"Using timing file path: {timing_file_path}")

        # Create a unique identifier for this query
        # A query is unique if it has the same text, mission group, and match count
        query_id = f"{query}|{mission_group or 'none'}|{requested_match_count}"

        # Initialize with empty data if file doesn't exist
        if not os.path.exists(timing_file_path):
            logger.info(f"Creating new timing file at {timing_file_path}")
            with open(timing_file_path, 'w') as f:
```

```
            with open(timing_file_path, 'w') as f:
                json.dump({"queries": []}, f, indent=2)

        # Read existing data with error handling
        try:
            with open(timing_file_path, 'r') as f:
                timing_data = json.load(f)

            # Validate the structure
            if not isinstance(timing_data, dict) or 'queries' not in timing_data:
                logger.warning("Invalid timing data structure, resetting file")
                timing_data = {"queries": []}
        except (json.JSONDecodeError, FileNotFoundError) as e:
            logger.error(f"Error reading timing file: {str(e)}")
            # Reset data if file is corrupted
            timing_data = {"queries": []}

        # Find if this query already exists
        existing_entry = None
        for entry in timing_data.get('queries', []):
            if (entry.get('query') == query and
                entry.get('mission_group') == mission_group and
                entry.get('match_count') == requested_match_count):
                existing_entry = entry
                break

        # Update existing entry or add new one
        if existing_entry:
            # Add timing to the existing entry
            if 'durations_ms' not in existing_entry:
                existing_entry['durations_ms'] = []
            if 'durations_sec' not in existing_entry:
                existing_entry['durations_sec'] = []

            existing_entry['durations_ms'].append(duration_ms_int)
            existing_entry['durations_sec'].append(duration_sec)
        else:
            # Create a new entry with a sequential ID
            new_id = len(timing_data.get('queries', [])) + 1
            new_entry = {
                'id': new_id,
                'query': query,
                'mission_group': mission_group,
                'match_count': requested_match_count,
                'durations_ms': [duration_ms_int],
                'durations_sec': [duration_sec]
            }
            timing_data['queries'].append(new_entry)

        # Write the updated data back to the file
        try:
            with open(timing_file_path, 'w') as f:
                json.dump(timing_data, f, indent=2)

            logger.info(f"Recorded timing for query: '{query}'")
            logger.info(f"Duration: {duration_ms_int} ms ({duration_sec} sec)")
        except Exception as write_error:
            logger.error(f"Error writing timing data: {str(write_error)}")
            return JsonResponse({'status': 'error', 'error': f'File write error: {str(write_error)}'}, status=500)

        return JsonResponse({'status': 'success'})

    except json.JSONDecodeError as e:
        logger.error(f"JSON decode error in timing data: {str(e)}")
        return JsonResponse({'status': 'error', 'error': f'Invalid JSON: {str(e)}'}, status=400)
    except Exception as e:
        logger.error(f"Error recording query timing: {str(e)}")
        logger.error(traceback.format_exc())
        return JsonResponse({'status': 'error', 'error': str(e)}, status=500)
```

M.2 Recorded time of processed queries


```
{
  "queries": [
    {
      "id": 1,
      "query": "How many postgraduates are in University of Cambridge in 2017?",
      "mission_group": null,
      "match_count": 2,
      "durations_ms": [
        6212,
        59,
        355      You, 6 days ago • uploading all the cleaning csv files nad final …
      ],
      "durations_sec": [
        6.21,
        0.06,
        0.36
      ]
    },
    {
      "id": 2,
      "query": "What was the research income starting in 2017?",
      "mission_group": null,
      "match_count": 2,
      "durations_ms": [
        5482,
        47
      ],
      "durations_sec": [
        5.48,
        0.05
      ]
    },
    {
      "id": 3,
      "query": "Show carbon emissions for Coventry University ending in 2021",
      "mission_group": null,
      "match_count": 1,
      "durations_ms": [
        13173
      ],
      "durations_sec": [
        13.17
      ]
    },
    {
      "id": 4,
      "query": "Show carbon emissions for Coventry University in 2021",
      "mission_group": null,
      "match_count": 1,
      "durations_ms": [
        4292,
        106
      ],
      "durations_sec": [
        4.29,
        0.11
      ]
```

```
  },
  {
    "id": 5,
    "query": "How many full-time students lived in university accommodation at University of York from 2016 to 2018?",
    "mission_group": null,
    "match_count": 4,
    "durations_ms": [
      8665,
      98
    ],
    "durations_sec": [
      8.67,
      0.1
    ]
  },
  {
    "id": 6,
    "query": "Research funding for University of Manchester from 2017 to 2019",
    "mission_group": null,
    "match_count": 3,
    "durations_ms": [
      10309,
      82
    ],
    "durations_sec": [
      10.31,
      0.08
    ]
  },
  {
    "id": 7,
    "query": "Graduation rates at London institutions for the past 5 years",
    "mission_group": null,
    "match_count": 5,
    "durations_ms": [
      12935
    ],
    "durations_sec": [
      12.94
    ]
  },
  {
    "id": 8,
    "query": "Graduation rates at University of London institutions for the past 5 years",
    "mission_group": null,
    "match_count": 5,
    "durations_ms": [
      58301,
      665,
      181
    ],
    "durations_sec": [
      58.3,
      0.67,
      0.18
    ]
  },
```