Evolution of

Phaidra Metadata & License Auditor

1. The Proof of Concept / Prototype Phase: Linear Execution

The first version of the code was a basic, hardcoded, linear script. It was designed to be modular (Acquisition, Analysis, Visualization) and to verify that the initial idea to retrieve/ acquire data via Phaidra's REST API and apply some basic visualization of the data is feasible:

- **Data Acquisition**
  - **Validating the API "Handshake":** I had to prove that the Phaidra API can actually talk to Python. If the API had been blocked or required a complex authentication, the project would have ended here.
  - **JSON Structure Discovery:** With the successful "Handshake" I could see the "shape" of the data. I would not the able to write the analysis logic (Phase 3) until I proved I could receive and process a JSON object and see where values like "mimetype" and "license" were hidden.
  - **Data Persistence (JSON):** I added a step to save the raw API response directly to a local .json file. It allows me to refine my analysis without needing to re-download the data every time, saving both time and server bandwidth.
  - **Testing the Library:** I tried several Python Libraries and was confident that the requests library was the right tool for the job. Once "Status 200" (Success) message came back, I knew I could dock onto the API.
- **Data Analysis**
  - **Basic Analysis:** I implemented a count function for "resourcetype"
- **Data Visualization**
  - **Basic Visualization:** Plots a basic diagram of the data.
- **Limitations:**
  - It could only retrieve the first 100 results.
  - Observation period was a fixed value (2023-2024)
  - Analysis Licenses data is missing entirely
- **Risk:** There was no error handling. Any out of the ordinary data or small input changes (hardcoded) could potentially crash the program.

2. The Scaling Phase: Pagination and Persistence

As the project grew, I needed more data for refinement, therefore the code evolved to handle large datasets through **pagination** and **local caching**.

- **Pagination:** The repositories APIs response can be massive. To segment the APIs response into smaller, more manageable packages, I implemented a while loop that compares the number of downloaded records to the totalFound value provided by the API. This allowed the program to fetch thousands of records in batches of 100.
- **Keyword Categorization:** I implemented a list of specific keywords to scan license strings. This allowed the program to automatically sort complex legal text into two clear categories: "Open Access License" (dc_license) or "All Rights reserved"

3. The Refinement Phase: Data Normalization

Once I had the raw data and observation period, the code had to handle the "messiness" of real-world metadata. This is where the logic became more sophisticated.

- **Data Cleaning and Fallback Logic:** By studying the JSON structure, I realized that file format is not always stored in the same data field, so I had to try different fields. If one field was empty, the code "fell back" to the next available one. The data sometimes is stored in different data types. To avoid the “unhashable type” - “List vs string” issue, a Fallback Logic was implemented.

4. The Architecture Phase: Interface and User Interaction

- **Function Interface:** I updated the fetch_all_phaidra_data to accept start_year and end_year as arguments.
- **User Interaction:** I use the input() function in the main block to ask for the years.
- **Opening Pandora’s Box:** By letting my wife test the input, she found a seemingly unlimited source of ways of how to torture me, by letting my program crash:
    - No data before 2008
    - Entering strings (not numbers)
    - Switching end_year and start_year
    - ...

5. The Reporting Phase: Input Validation, Visualization and Export

The final evolutionary step was Validating turning internal data into human-readable external reports.

- **Input Validation:** Restricts certain User Inputs
- **Tabular Export:** I added the csv module to generate spreadsheets that can be opened in Excel.

- **Data Visualization:** I integrated matplotlib and numpy to convert raw numbers into bar charts. This allowed the program to communicate the "Open Access" status of the repository visually rather than just numerically.