Report: Python Information Aggregator with Web API and Scraping

Introduction

This report documents the development of a Python-based Information Aggregator that combines data from web APIs and web scraping techniques. The program fetches news articles from various sources, processes the data, and presents it through a graphical user interface (GUI). The implementation adheres to object-oriented programming (OOP) principles and includes unit testing for reliability. This report outlines the challenges faced, design decisions made, and additional features implemented.

Key Components Implemented

1. News API Integration

The program integrates with the News API to fetch current news headlines and articles. Users can select their preferred news sources or filter articles by category and country. The API key management system ensures valid keys are used, cycling through multiple keys if necessary to avoid rate limits.

2. Web Scraping

For sources not fully covered by the API (e.g., ABC Australia), the program uses BeautifulSoup to scrape additional details such as article content, authors, and publication dates. This enriches the dataset beyond what the API provides.

3. Data Processing

The program combines data from both API responses and web scraping into a unified pandas DataFrame. It handles data cleaning, removes duplicates, and ensures consistency in the extracted information.

4. Data Visualization

A bar chart is generated to visualize the most common words in article titles, providing insights into trending topics. The visualization is embedded in the GUI using Matplotlib.

5. Object-Oriented Programming (OOP) Principles

The code is structured using OOP principles:

* **Encapsulation**: The KeyList and NewsAPI\_KeyList classes manage API keys securely.
* **Inheritance**: NewsAPI\_KeyList inherits from KeyList to extend functionality.
* **Modularity**: Separate classes and methods handle API calls, scraping, and GUI components.

6. Unit Testing

Critical functions, such as API key validation and data extraction, are tested using the unittest framework. This ensures reliability and correctness.

7. Graphical User Interface (GUI)

The GUI, built with Tkinter, allows users to:

* Select news sources, categories, and countries.
* Fetch and display articles.
* View visualizations of common title words.

Challenges Faced

1. API Key Management

Handling multiple API keys and ensuring their validity was challenging. The solution involved creating a key management system that cycles through keys and checks their validity before use.We had an API limit of 100 so we created multiple API keys with many Email addresses.

2. Web Scraping Dynamic Content

Scraping dynamically loaded content from news websites (e.g., ABC Australia) required parsing JSON-LD structured data embedded in scripts. This was addressed by extracting and processing JSON data directly.

3. Data Consistency

Combining data from APIs and scraping introduced inconsistencies in date formats and missing fields. The program standardizes dates and handles missing values gracefully.

4. Performance

Fetching and rendering large datasets impacted performance. Optimizing DataFrame operations improved responsiveness.

Design Decisions

1. Class Structure

* KeyList**and**NewsAPI\_KeyList: Centralized API key management.
* NewsScraper: Combines API calls, scraping, and GUI logic for a cohesive user experience.

2. User Interaction

* Radio buttons toggle between source-based and category/country-based filtering.
* Listboxes allow multi-selection for flexibility.

3. Visualization

* Matplotlib was chosen for its simplicity and integration with Tkinter.
* Common words are displayed in a bar chart for quick trend analysis.

Additional Features

1. Dynamic Key Validation

The program validates API keys in real-time, ensuring only working keys are used.

2. Error Handling

Comprehensive error handling prevents crashes during API failures or scraping errors.

3. Optional Features

* Users can limit the number of articles fetched (via pageSize).
* The GUI includes clear feedback for user actions.

Conclusion

The Information Aggregator successfully combines API data and web scraping to provide a comprehensive news aggregation tool. The use of OOP principles, unit testing, and a user-friendly GUI ensures the program is robust, maintainable, and accessible. Future enhancements could include caching, more advanced visualizations, and support for additional news sources.

Ethical Considerations

* The program respects API rate limits and terms of service.
* Web scraping is performed ethically, targeting only publicly available data.
* Users are reminded to use the tool responsibly and comply with legal guidelines.

Submission Requirements Met

1. **Codebase**: Well-documented Python code adhering to OOP principles.
2. **README**: Instructions for running the program, API key setup, and dependencies.
3. **Report**: This document covers challenges, design decisions, and features.
4. **GUI Screenshots**: Included in the submission folder.
5. **Video Presentation**: A separate video demonstrates the tool's functionality