36122 PYTHON PROGRAMMING
TD School
University of Technology Sydney

Information Aggregator with Web API and Scraping

Student Name: Shaqran Bin Saleh
Student ID: 25010238

Student Name:

Student ID:

Student Name:

Student ID:

Student Name:

Student ID:

Student Name:

Student ID:

Table of Contents

1. Introduction

In today’s data-driven world, the ability to retrieve, enrich, and present information from various online sources is a critical skill. The assignment titled "Python Information Aggregator with Web API and Scraping" is designed to provide hands-on experience in building a robust and interactive data collection and analysis tool. This tool fetches news articles using the NewsAPI and augments the retrieved data using web scraping techniques. The enriched dataset is then visualized and displayed through a Graphical User Interface (GUI), enabling users to interact with the data in a meaningful and intuitive way.

The assignment encapsulates essential skills in API integration, web scraping, object-oriented programming (OOP), GUI development, and data visualization. Furthermore, it emphasizes software quality through unit testing and efficient data management using JSON-based caching. By combining multiple facets of Python programming, the project not only deepens the student’s technical knowledge but also simulates the development of a real-world application. The resulting system is capable of delivering a topic-agnostic and user-customizable news aggregation platform, empowering users to explore, analyze, and interact with real-time information using a clean and intuitive interface.

This project offers a multidisciplinary approach by integrating software engineering principles, user experience design, and data science methodologies. It stands as a demonstration of practical problem-solving using Python's extensive ecosystem, preparing students for professional environments where automation, data analytics, and usability converge. The assignment not only meets the academic goals of teaching theoretical concepts but also serves as a mini-capstone project illustrating full-cycle software development.

2. Objective Overview

The main objective of this assignment is to develop a flexible and user-friendly Python-based application that aggregates and enriches information from online sources. This is achieved through four primary goals:

a. Versatile Information Aggregator

The project demonstrates how to design a topic-agnostic aggregator that can pull relevant data from an external API. In this case, the NewsAPI is used to gather the latest articles from various categories including business, health, entertainment, and technology. By allowing users to select categories and control the number of articles retrieved, the aggregator ensures adaptability to a wide range of user interests and information needs.

Moreover, the design of the aggregator ensures scalability and modularity, allowing future expansion into other APIs or domains such as weather, finance, or sports. The application architecture is open-ended, supporting pluggable components that can be extended with minimal changes to the core logic.

b. Object-Oriented Programming (OOP) Principles

The application is built using a modular, object-oriented structure. Key classes such as `Article`, `NewsFetcher`, `WebScraper`, and `NewsAggregator` ensure code clarity, reusability, and scalability. Encapsulation of logic within these classes simplifies debugging and future enhancements. This approach also allows students to gain hands-on experience in applying core programming paradigms that are vital for building maintainable and professional software.

OOP principles help promote separation of concerns and simplify unit testing. By keeping functionality isolated, the code becomes less prone to side effects and easier to maintain. The use of well-named methods and clean interfaces between classes also aligns with software engineering best practices.

c. Reliable and Tested Code

To ensure the reliability and maintainability of the code, unit testing is implemented using Python's `unittest` framework. This allows systematic validation of key components such as data fetching, scraping, and deduplication. Unit testing not only helps catch errors early but also builds confidence in the software's correctness and performance across different inputs and scenarios.

Robust unit tests serve as both a safety net and documentation. They enable future refactoring by providing assurance that new changes do not break existing functionality. This promotes a culture of test-driven development (TDD) and continuous integration.

d. User-Friendly Graphical User Interface (GUI)

A core objective is to provide a rich user experience through a Tkinter-based GUI. Users can select the news category, choose the number of articles to fetch, and visualize results directly within the application. The GUI transforms the backend functionality into an accessible and interactive front end, catering to users who may not be familiar with code execution environments or command-line interfaces.

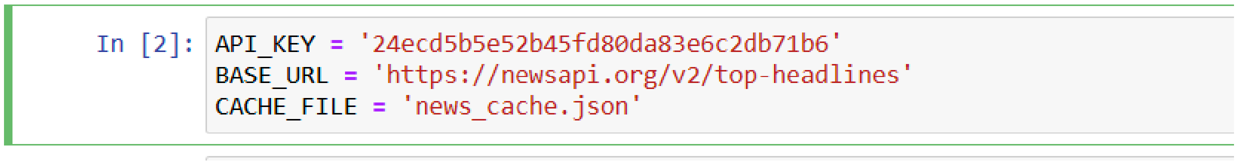
The inclusion of a GUI increases the reach of the application, making it usable in classrooms, homes, and workplaces. Its responsive design and error-handling mechanisms make the system resilient to invalid inputs and unexpected API failures.

3. Key Components

A. API Integration

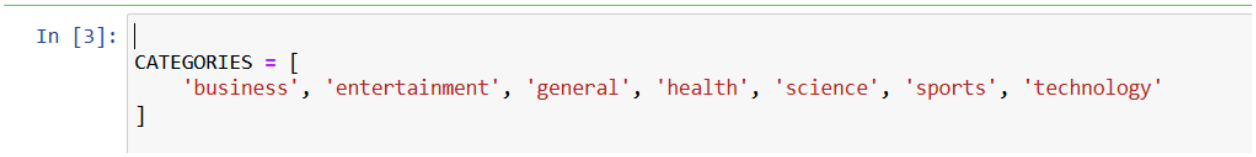
1. Importance of Selecting a Relevant Public API

The selection of a suitable API is crucial for meaningful data aggregation. The NewsAPI is chosen for its broad access to real-time headlines across multiple topics and sources. Its JSON response format is easy to parse and includes metadata like title, author, publish date, and URL.



```
In [2]: API_KEY = '24ecd5b5e52b45fd80da83e6c2db71b6'
        BASE_URL = 'https://newsapi.org/v2/top-headlines'
        CACHE_FILE = 'news_cache.json'
```

Furthermore, NewsAPI provides filtering options such as category, country(we have considered only US), and source, which make it ideal for building a versatile and responsive aggregator.

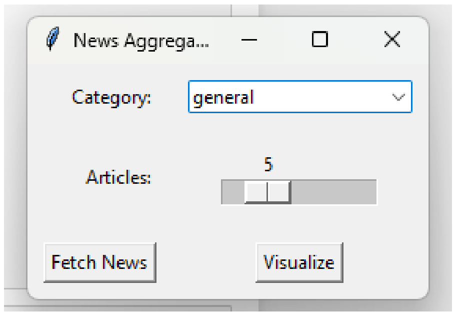


```
In [3]: 
        CATEGORIES = [
            'business', 'entertainment', 'general', 'health', 'science', 'sports', 'technology'
        ]
```

Choosing a reliable public API ensures the system can maintain high uptime and offer real-time data that reflects current events. This real-world relevance enriches the educational value of the assignment and creates practical awareness of the challenges associated with API limitations, such as rate limits, response delays, and authentication.

2. User Input for Data Retrieval

The GUI allows users to dynamically select news categories and adjust the number of articles to fetch using a dropdown menu and a slider widget. These parameters are passed to the API call, ensuring flexibility and user control over the data retrieval process. This empowers users to tailor their experience based on personal interests, whether they seek trending news in technology or developments in health and science.



Category: general

Articles: 5

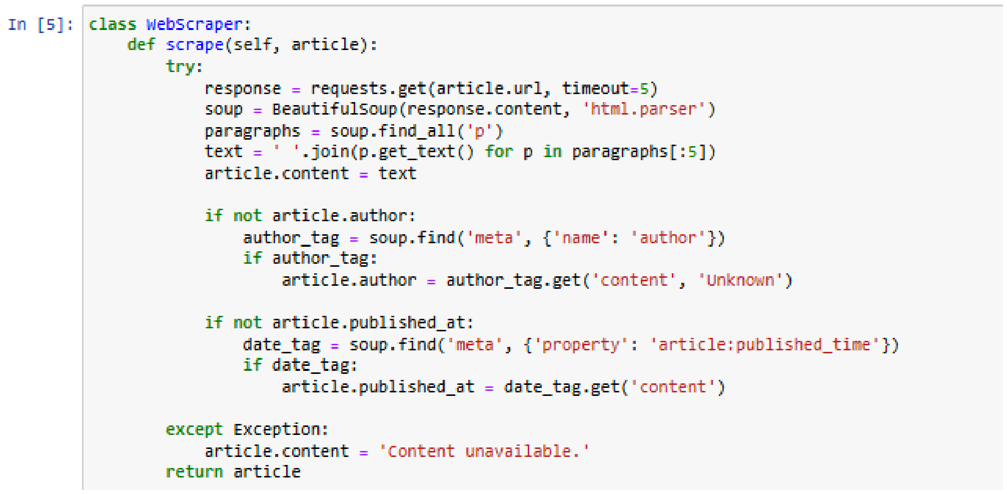
```python
class NewsAppGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("News Aggregator")
        self.aggregator = NewsAggregator()

        tk.Label(root, text="Category:").grid(row=0, column=0, padx=10, pady=10, sticky='e')
        self.category_var = tk.StringVar(value='general')
        self.category_menu = ttk.Combobox(root, textvariable=self.category_var, values=CATEGORIES)
        self.category_menu.grid(row=0, column=1, padx=10, pady=10)

        tk.Label(root, text="Articles:").grid(row=1, column=0, padx=10, pady=10, sticky='e')
        self.article_slider = tk.Scale(root, from_=1, to=20, orient='horizontal')
        self.article_slider.set(5)
        self.article_slider.grid(row=1, column=1, padx=10, pady=10)

        self.fetch_btn = tk.Button(root, text="Fetch News", command=self.fetch_news)
        self.fetch_btn.grid(row=2, column=0, padx=10, pady=10)

        self.visual_btn = tk.Button(root, text="Visualize", command=self.visualize)
        self.visual_btn.grid(row=2, column=1, padx=10, pady=10)

    def fetch_news(self):
        try:
            category = self.category_var.get()
            limit = self.article_slider.get()
            articles = self.aggregator.get_articles(category, limit)
            df = self.aggregator.to_dataframe(articles)
            df_view = df[['title', 'author', 'source', 'published_at', 'url']]

            table_window = tk.Toplevel(self.root)
            table_window.title("News Articles")

            columns = ('Title', 'Author', 'Source', 'Published At', 'URL')
            tree = ttk.Treeview(table_window, columns=columns, show='headings')

            for col in columns:
                tree.heading(col, text=col)
                tree.column(col, anchor='w', width=200)

            for _, row in df_view.iterrows():
                tree.insert('', 'end', values=(
                    row['title'], row['author'], row['source'], row['published_at'], row['url']
                ))

            tree.pack(fill='both', expand=True)

            scrollbar = ttk.Scrollbar(table_window, orient='vertical', command=tree.yview)
            tree.configure(yscroll=scrollbar.set)
            scrollbar.pack(side='right', fill='y')

            # Make URL clickable on double-click
            def open_link(event):
                selected = tree.focus()
                if selected:
                    values = tree.item(selected, 'values')
                    url = values[4]
                    if url.startswith('http'):
                        webbrowser.open_new_tab(url)

            tree.bind('<Double-1>', open_link)

        except Exception as e:
            messagebox.showerror("Error", str(e))

    def visualize(self):
        try:
            category = self.category_var.get()
            limit = self.article_slider.get()
            articles = self.aggregator.get_articles(category, limit)
            self.aggregator.visualize(articles)
        except Exception as e:
            messagebox.showerror("Error", str(e))
```

User input adds a layer of customization and interactivity, enhancing the application's utility across different user profiles. It also introduces the concept of parameterized data requests, which is fundamental to many real-world data pipelines.

B. Web Scraping for Enhanced Information

1. Significance of Web Scraping

APIs often provide limited data to reduce payload size and complexity. To enrich the article content, web scraping is used to fetch the actual content of each article by visiting the URL. This provides deeper insights by including full text, authorship, and publish time if missing. Scraping allows the application to overcome the limitations of third-party APIs and deliver more comprehensive and contextual information to users.

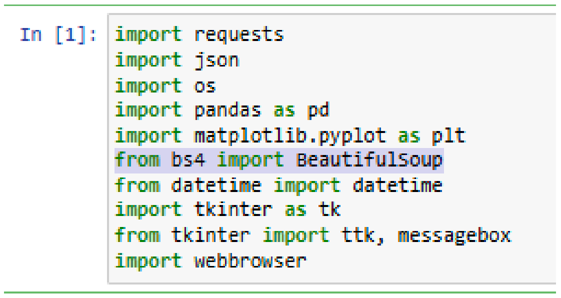


```
In [5]: class WebScraper:
            def scrape(self, article):
                try:
                    response = requests.get(article.url, timeout=5)
                    soup = BeautifulSoup(response.content, 'html.parser')
                    paragraphs = soup.find_all('p')
                    text = ' '.join(p.get_text() for p in paragraphs[:5])
                    article.content = text

                    if not article.author:
                        author_tag = soup.find('meta', {'name': 'author'})
                        if author_tag:
                            article.author = author_tag.get('content', 'Unknown')

                    if not article.published_at:
                        date_tag = soup.find('meta', {'property': 'article:published_time'})
                        if date_tag:
                            article.published_at = date_tag.get('content')

                except Exception:
                    article.content = 'Content unavailable.'
                return article
```

This enrichment process is especially useful when building dashboards or analytical models that require full textual content. It reflects the common industry practice of combining structured API data with unstructured web content.

2. Libraries Used for Web Scraping

BeautifulSoup, part of the `bs4` library, is employed for HTML parsing. It enables searching and extracting specific elements such as paragraphs and meta tags. This library is both flexible and easy to use, making it a popular choice for web scraping tasks. The use of BeautifulSoup ensures that scraping operations are fast, readable, and adaptable to changes in webpage structure.



```
In [1]: import requests
        import json
        import os
        import pandas as pd
        import matplotlib.pyplot as plt
        from bs4 import BeautifulSoup
        from datetime import datetime
        import tkinter as tk
        from tkinter import ttk, messagebox
        import webbrowser
```

In advanced scenarios, developers might also combine BeautifulSoup with other tools like `requests`, `lxml`, or even headless browsers for scraping JavaScript-generated content. However, for this assignment, BeautifulSoup suffices for efficiently gathering readable content from traditional HTML-based pages.

C. Object-Oriented Programming (OOP) Principles

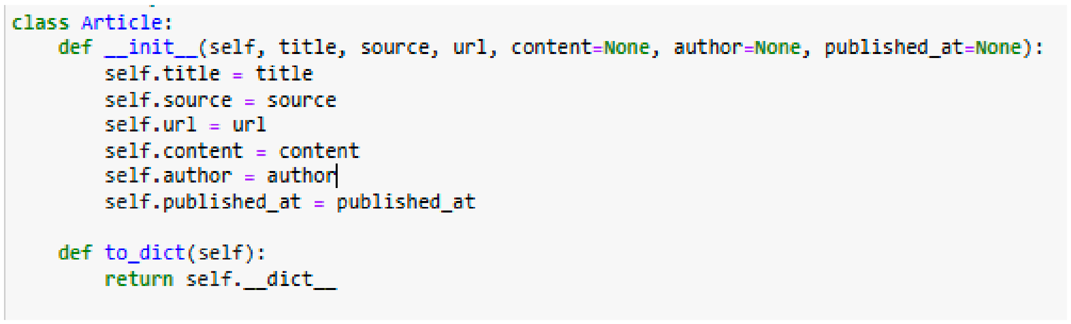
1. Benefits of OOP in Code Design

OOP facilitates modular and maintainable code. Each component of the application is encapsulated within a class that has clearly defined responsibilities. This design pattern supports easy debugging, testing, and future upgrades. OOP also makes the codebase easier to understand and modify, especially when collaborating in teams or integrating with other projects.

Well-structured object-oriented code often results in lower technical debt, faster onboarding for new developers, and better adaptability to shifting requirements. It provides a shared language and structure for expressing business logic and user interactions.

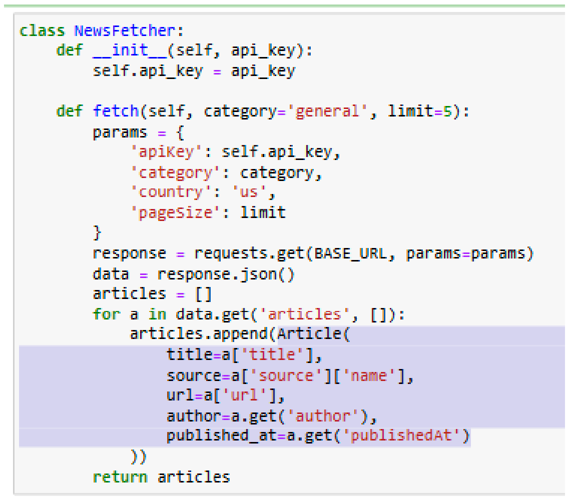
2. Encapsulation, Inheritance, and Polymorphism

- **Encapsulation**: Each class, such as `Article`, encapsulates its data and behaviors. This prevents external modification of internal states and promotes data integrity.

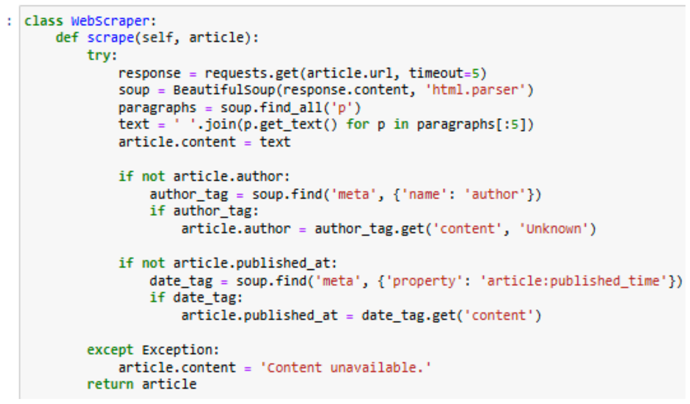


```
class Article:
    def __init__(self, title, source, url, content=None, author=None, published_at=None):
        self.title = title
        self.source = source
        self.url = url
        self.content = content
        self.author = author
        self.published_at = published_at

    def to_dict(self):
        return self.__dict__
```

- **Inheritance**: While inheritance is not directly used, the structure is open to extending classes like `NewsFetcher` or `WebScraper`. For instance, subclasses could be created for fetching articles from different APIs with minimal changes to the rest of the application.



```python
class NewsFetcher:
    def __init__(self, api_key):
        self.api_key = api_key

    def fetch(self, category='general', limit=5):
        params = {
            'apiKey': self.api_key,
            'category': category,
            'country': 'us',
            'pageSize': limit
        }
        response = requests.get(BASE_URL, params=params)
        data = response.json()
        articles = []
        for a in data.get('articles', []):
            articles.append(Article(
                title=a['title'],
                source=a['source']['name'],
                url=a['url'],
                author=a.get('author'),
                published_at=a.get('publishedAt')
            ))
        return articles
```



```python
class WebScraper:
    def scrape(self, article):
        try:
            response = requests.get(article.url, timeout=5)
            soup = BeautifulSoup(response.content, 'html.parser')
            paragraphs = soup.find_all('p')
            text = ' '.join(p.get_text() for p in paragraphs[:5])
            article.content = text

            if not article.author:
                author_tag = soup.find('meta', {'name': 'author'})
                if author_tag:
                    article.author = author_tag.get('content', 'Unknown')

            if not article.published_at:
                date_tag = soup.find('meta', {'property': 'article:published_time'})
                if date_tag:
                    article.published_at = date_tag.get('content')

        except Exception:
            article.content = 'Content unavailable.'
        return article
```

- **Polymorphism**: The interface for scraping or visualizing can be modified to work with different data types, simulating polymorphic behavior. This allows different modules to interact with common methods despite underlying differences in data sources.

These principles are foundational for building extensible systems and are considered best practices in both academic and professional software development environments.

D. Unit Testing

1. Importance of Unit Testing

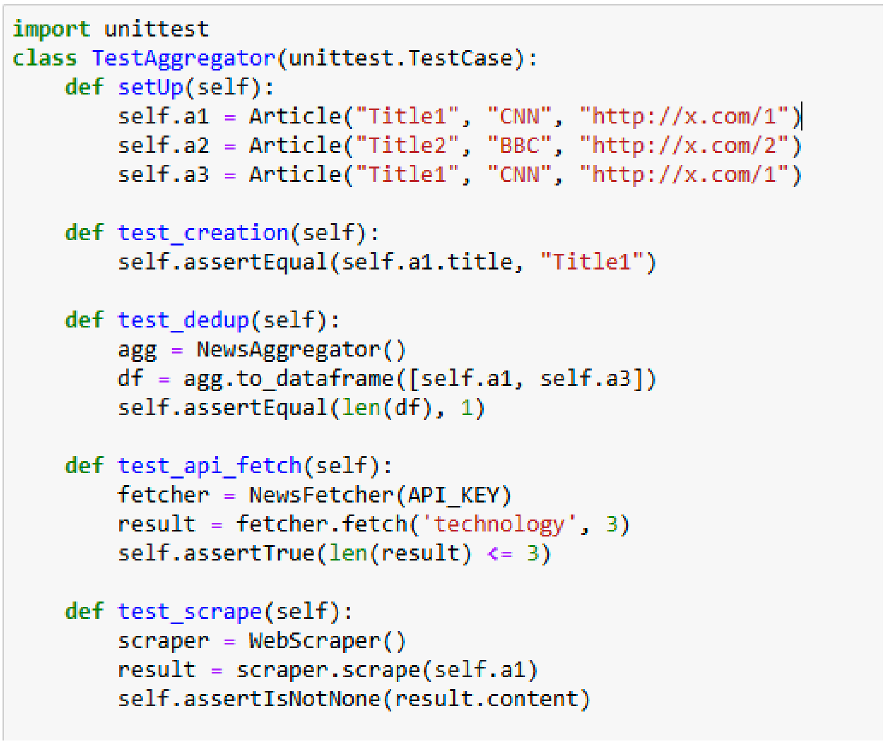
Testing ensures the application performs as expected. Unit tests help identify bugs early, verify new features, and protect existing functionality from regressions. By incorporating testing into the development lifecycle, the application becomes more robust and less prone to unexpected failures.

In production systems, automated testing is indispensable for continuous delivery and deployment pipelines. It also aids in documentation by clarifying the intended behavior of each function or class.

2. Testing Framework Used

The built-in `unittest` module is used to test critical functions:

- Article object creation
- Deduplication logic
- API response constraints
- Scraping functionality



```python
import unittest
class TestAggregator(unittest.TestCase):
    def setUp(self):
        self.a1 = Article("Title1", "CNN", "http://x.com/1")
        self.a2 = Article("Title2", "BBC", "http://x.com/2")
        self.a3 = Article("Title1", "CNN", "http://x.com/1")

    def test_creation(self):
        self.assertEqual(self.a1.title, "Title1")

    def test_dedup(self):
        agg = NewsAggregator()
        df = agg.to_dataframe([self.a1, self.a3])
        self.assertEqual(len(df), 1)

    def test_api_fetch(self):
        fetcher = NewsFetcher(API_KEY)
        result = fetcher.fetch('technology', 3)
        self.assertTrue(len(result) <= 3)

    def test_scrape(self):
        scraper = WebScraper()
        result = scraper.scrape(self.a1)
        self.assertIsNotNone(result.content)
```

These tests provide a layer of confidence that the system functions correctly. Additionally, they serve as documentation for the expected behavior of each component, which is especially useful for future developers and maintainers.

The modular nature of the code makes it ideal for unit testing. By isolating logic into discrete classes and methods, each test case can focus on a single responsibility, following the Arrange-Act-Assert pattern.

E. User Interaction with GUI

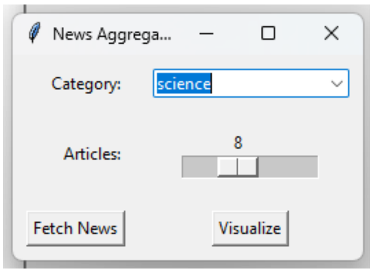
1. Introduction of GUI for Enhanced UX

A Tkinter-based GUI makes the application accessible to non-technical users. It abstracts the code complexity and presents an intuitive interface for selecting news categories, viewing articles, and visualizing trends. This significantly lowers the barrier to entry and makes the application suitable for a broader audience.

Interactive applications not only improve usability but also increase engagement, making the learning process more effective for students. It teaches the importance of human-centered design in software applications.

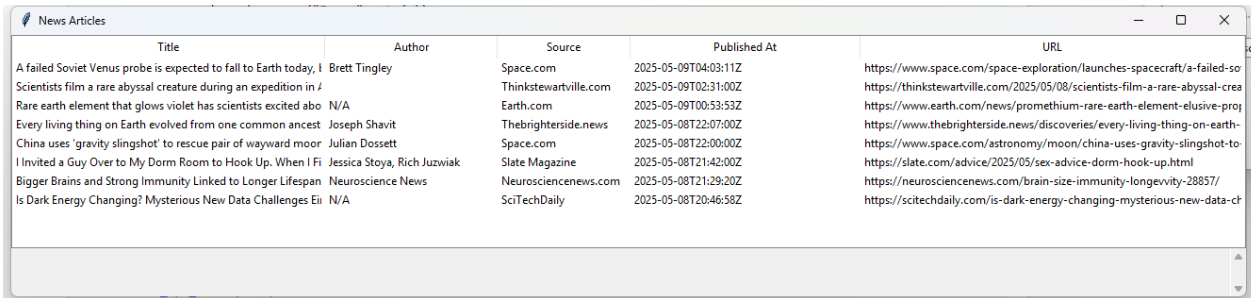
2. GUI Libraries Used

The `tkinter` module, along with `ttk` for advanced widgets, builds the user interface. Message pop-ups (`messagebox`) handle exceptions and provide user feedback, while `webbrowser` is used to open article URLs with a double-click. These tools together create a smooth and interactive experience that mirrors the usability of commercial desktop applications.

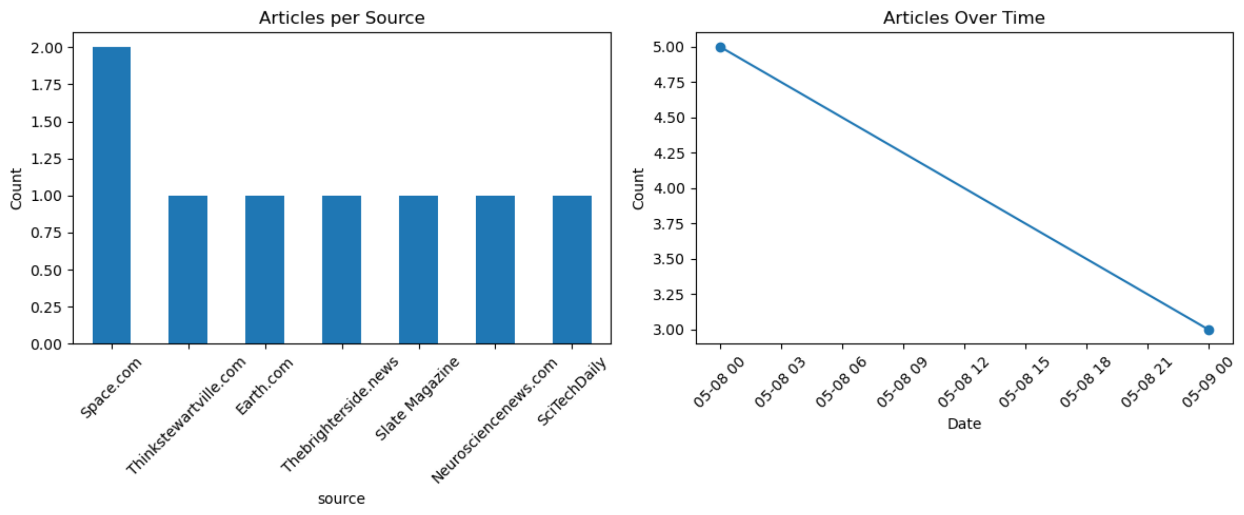


Category: science

Articles: 8



Title	Author	Source	Published At	URL
A failed Soviet Venus probe is expected to fall to Earth today, I	Brett Tingley	Space.com	2025-05-09T04:03:11Z	https://www.space.com/space-exploration/launches-spacecraft/a-failed-so
Scientists film a rare abyssal creature during an expedition in A		Thinkstewartville.com	2025-05-09T02:31:00Z	https://thinkstewartville.com/2025/05/08/scientists-film-a-rare-abyssal-crea
Rare earth element that glows violet has scientists excited abo	N/A	Earth.com	2025-05-09T00:53:53Z	https://www.earth.com/news/promethium-rare-earth-element-elusive-pro
Every living thing on Earth evolved from one common ancest	Joseph Shavit	Thebrighterside.news	2025-05-08T22:07:00Z	https://www.thebrighterside.news/discoveries/every-living-thing-on-earth-
China uses 'gravity slingshot' to rescue pair of wayward moor	Julian Dossett	Space.com	2025-05-08T22:00:00Z	https://www.space.com/astronomy/moon/china-uses-gravity-slingshot-to-
I Invited a Guy Over to My Dorm Room to Hook Up. When I Fi	Jessica Stoya, Rich Juzwiak	Slate Magazine	2025-05-08T21:42:00Z	https://slate.com/advice/2025/05/sex-advice-dorm-hook-up.html
Bigger Brains and Strong Immunity Linked to Longer Lifespan	Neuroscience News	Neurosciencenews.com	2025-05-08T21:29:20Z	https://neurosciencenews.com/brain-size-immunity-longevvity-28857/
Is Dark Energy Changing? Mysterious New Data Challenges Ei	N/A	SciTechDaily	2025-05-08T20:46:58Z	https://scitechdaily.com/is-dark-energy-changing-mysterious-new-data-ch



Tkinter is a built-in library, making it easy to deploy without external dependencies. It also provides a gateway for learning more advanced GUI frameworks like PyQt, Kivy, or ElectronJS.

F. Optional Features and Data Visualization

1. Optional Features: Caching and Article Limits

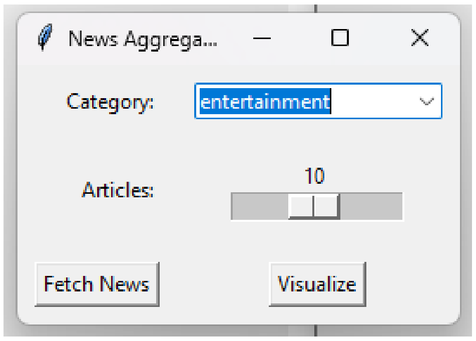
Caching is implemented using a JSON file to reduce API calls and improve performance. Each API response is stored with a key (e.g., "business:5") and retrieved later if available. This not only saves bandwidth but also accelerates response time for repeated queries. The user-defined article limit feature enables tailored content delivery, offering scalability for light or in-depth reading.

Such optimizations reflect real-world concerns like cost minimization (e.g., API pricing), latency reduction, and offline access.

2. Data Visualization with Matplotlib

The application features two types of plots:

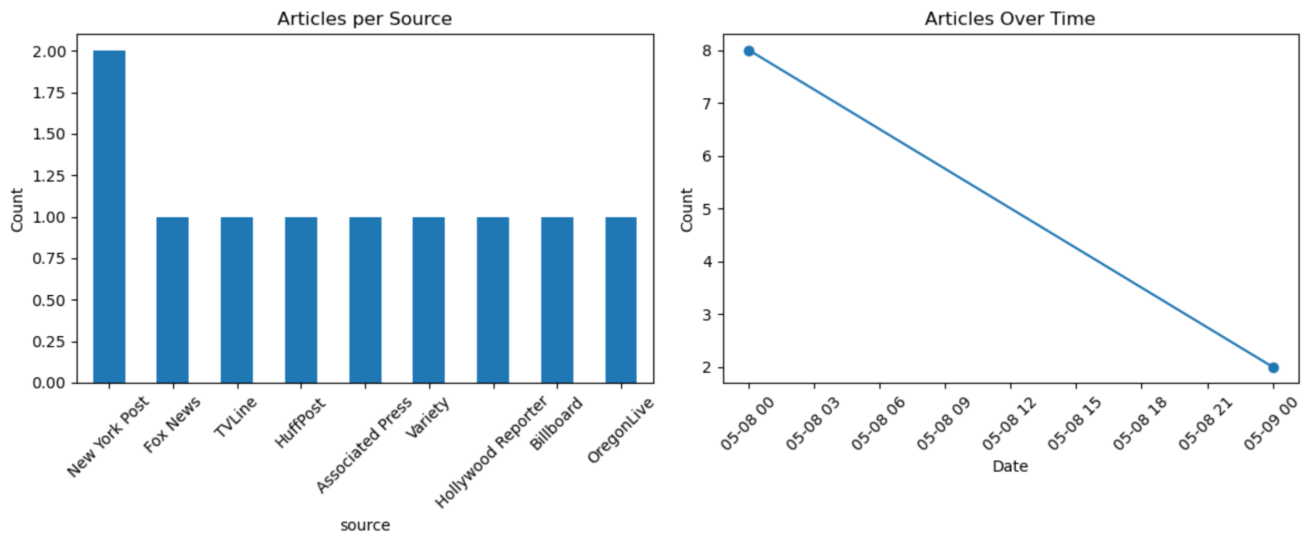
- **Bar chart** showing the number of articles per source.
- **Line chart** showing article publishing trends over time.



News Aggrega...

Category: entertainment

Articles: 10



These visual insights help users understand patterns and source bias in the news dataset. Matplotlib, a widely-used plotting library, ensures high-quality and customizable visualizations. With additional features, these plots could be extended to include sentiment analysis, keyword frequency, or geographic distribution.

Visualization plays a key role in turning raw data into actionable insights. It enhances decision-making, storytelling, and engagement in academic or business contexts.

4. Recommendations

While the project meets its initial objectives effectively, there are several areas where improvements and additional features could significantly enhance the system's capabilities and user experience:

1. **Multi-Country News Support**:
   Extend the NewsAPI query parameters to allow users to select news based on different countries. This would broaden the scope and diversity of the articles presented, making the application suitable for a more global audience.
2. **Search Functionality by Keyword**:
   Introduce a text entry widget in the GUI that allows users to input keywords (e.g., "climate change," "elections," or "AI"). This would enable more targeted searches and allow for deeper exploration of topics.
3. **Theme Customization and Accessibility**:
   Allow users to switch between light and dark themes or adjust font sizes. These accessibility features can make the application more comfortable for prolonged use.
4. **Exporting Data**:
   Include options to export fetched articles to CSV or PDF for academic, professional, or personal use. This would make the application more versatile and functional.

5. **Mobile and Web Interface**:
   As a stretch goal, the application could be re-developed using a web framework (e.g., Flask, Django) or mobile SDK (e.g., Kivy or React Native) to make it accessible across platforms.
6. **Sentiment Analysis and NLP**:
   Integrate a basic natural language processing (NLP) module using libraries like NLTK or TextBlob to detect sentiment or summarize article content. This adds analytical value to the application.
7. **Database Integration**:
   Transition from JSON to SQLite or another lightweight database system for structured and persistent data storage. This supports more complex queries and better data organization.

These enhancements can elevate the project from a functional prototype to a professional-grade product with broad utility in education, journalism, and data science.

5. Conclusion

The "Python Information Aggregator with Web API and Scraping" project effectively demonstrates the integration of several core components of software development. By combining API integration, web scraping, OOP principles, unit testing, GUI interaction, and data visualization, the assignment showcases how a well-structured Python application can deliver real-time, personalized, and enriched information to users.

From a pedagogical perspective, the project enables students to apply theoretical knowledge in a practical context, preparing them for challenges faced in real-world development environments. The system’s modular design and emphasis on software engineering best practices, including testing and caching, make it not only functional but also scalable and maintainable.

The addition of user interactivity via a GUI ensures that the application is accessible and engaging. Meanwhile, the implementation of data enrichment through scraping and visualization through Matplotlib transforms static data into a dynamic experience.

In conclusion, this project provides a comprehensive learning platform and a practical blueprint for building intelligent information systems. It sets the stage for future development in areas like advanced analytics, cross-platform deployment, and user-centered design. With further refinements, this tool can evolve from an academic exercise into a full-featured product suitable for end users in a variety of domains.

6. References

• NewsAPI. (n.d.). *News API - A JSON-based API for live news and blog articles*. Retrieved from https://newsapi.org

• Beautiful Soup Documentation. (n.d.). *Beautiful Soup: We called him Tortoise because he taught us*. Retrieved from https://www.crummy.com/software/BeautifulSoup/bs4/doc/

• Matplotlib Developers. (n.d.). *Matplotlib: Visualization with Python*. Retrieved from https://matplotlib.org

• Python Software Foundation. (n.d.). *unittest — Unit testing framework*. Retrieved from https://docs.python.org/3/library/unittest.html

• TkDocs. (n.d.). *Tkinter 8.5 reference: a GUI for Python*. Retrieved from https://tkdocs.com/

• pandas Development Team. (n.d.). *pandas: Python Data Analysis Library*. Retrieved from https://pandas.pydata.org

• Python Software Foundation. (n.d.). *The Python Standard Library*. Retrieved from https://docs.python.org/3/library/