Name:-Rion Tuscano
Rollno:- 10464
Div:-TEComps-A
Lab:- OSINT

Project Report: Automated Social Media OSINT Aggregation Pipeline

1. Introduction

Open-Source Intelligence (OSINT) is the practice of collecting and analyzing data from publicly available sources to produce actionable intelligence. In the digital age, social media platforms have become vast reservoirs of public sentiment, breaking news, and emerging trends. The manual collection of this data is inefficient and unscalable. This project addresses this challenge by creating an automated pipeline to aggregate OSINT from various social media platforms.

The primary objective of this lab was to design, build, and deploy a modular and automated OSINT pipeline using Python. The key goals included:

- Utilizing social media APIs to programmatically collect data from diverse sources like Twitter, Reddit, and Facebook.
- Structuring a robust Python pipeline to handle the collection, preprocessing, and storage of the aggregated data.
- Enriching the collected data through processes such as text cleaning, language detection, and sentiment analysis to enable cross-platform intelligence analysis.

This report details the methodology employed to construct the pipeline, presents the results obtained, discusses the challenges encountered, and proposes future enhancements.

2. Methodology

The project was executed by developing a modular pipeline with distinct components for data collection, processing, storage, and visualization. The architecture was designed to be easily extensible, allowing for the addition of new data sources with minimal effort.

2.1. Integrated Platforms

The pipeline was engineered to collect data from a wide array of social media and online platforms to ensure comprehensive coverage. The integrated sources include:

- Twitter (via tweepy and snscrape)
- Reddit (via praw)
- Facebook (via rapidapi)
- Instagram (via rapidapi)
- GitHub (via github)
- TikTok (via rapidapi)
- VK (via rapidapi)

2.2. Tools and Libraries

A collection of specialized Python libraries was used to build the pipeline. These tools were chosen for their specific capabilities in API interaction, data manipulation, and automation

- **API Access & Scraping:** tweepy, praw, facebook-sdk, instaloader, snscrape, TikTokApi, linkedin-api, telethon, discord.py, Mastodon.py, PyGithub, vk-api, requests, beautifulsoup
- **Data Processing & Analysis:** pandas, numpy, textblob, langdetect, nltk, scikit-learn.
- **Database Management:** sqlalchemy and the built-in sqlite3 module were used for database operations.
- **Automation & Configuration:** schedule was used for task automation, and python-dotenv for securely managing API credentials.
- **Visualization:** matplotlib was used for generating charts.

2.3. Pipeline Architecture

The pipeline followed a structured, multi-stage process from data ingestion to storage:

1. **Configuration:** API keys and secrets were stored securely in a .env file and loaded into the application at runtime.

2. **Collection:** Modular collector scripts (e.g., twitter_collector.py, reddit_collector.py) were executed to fetch data from their respective platforms. Each collector was designed to return data in a unified schema.

3. **Cleaning & Preprocessing:** The raw data was passed through a cleaning utility (cleaner.py). This stage involved removing URLs and special characters from the text and filtering out non-English records usinglangdetect.

4. **Enrichment:** The cleaned data was enriched with sentiment analysis scores. Using TextBlob, a polarity score (ranging from -1.0 to 1.0) was calculated for each text record and appended to it.

5. **Storage:** The processed and enriched records were saved to a persistent SQLite database (data/osint.db). A dedicated table,osint_data, was created to store the information according to the defined schema.

6. **Automation:** The entire pipeline was wrapped in a function (run_pipeline) and scheduled to run automatically every hour using the schedule library, ensuring continuous data collection.

3. Results

The pipeline successfully collected, processed, and stored over 100 OSINT records from the configured platforms. The unified schema ensured that data from disparate sources could be analyzed collectively.

3.1. Sample Database Records

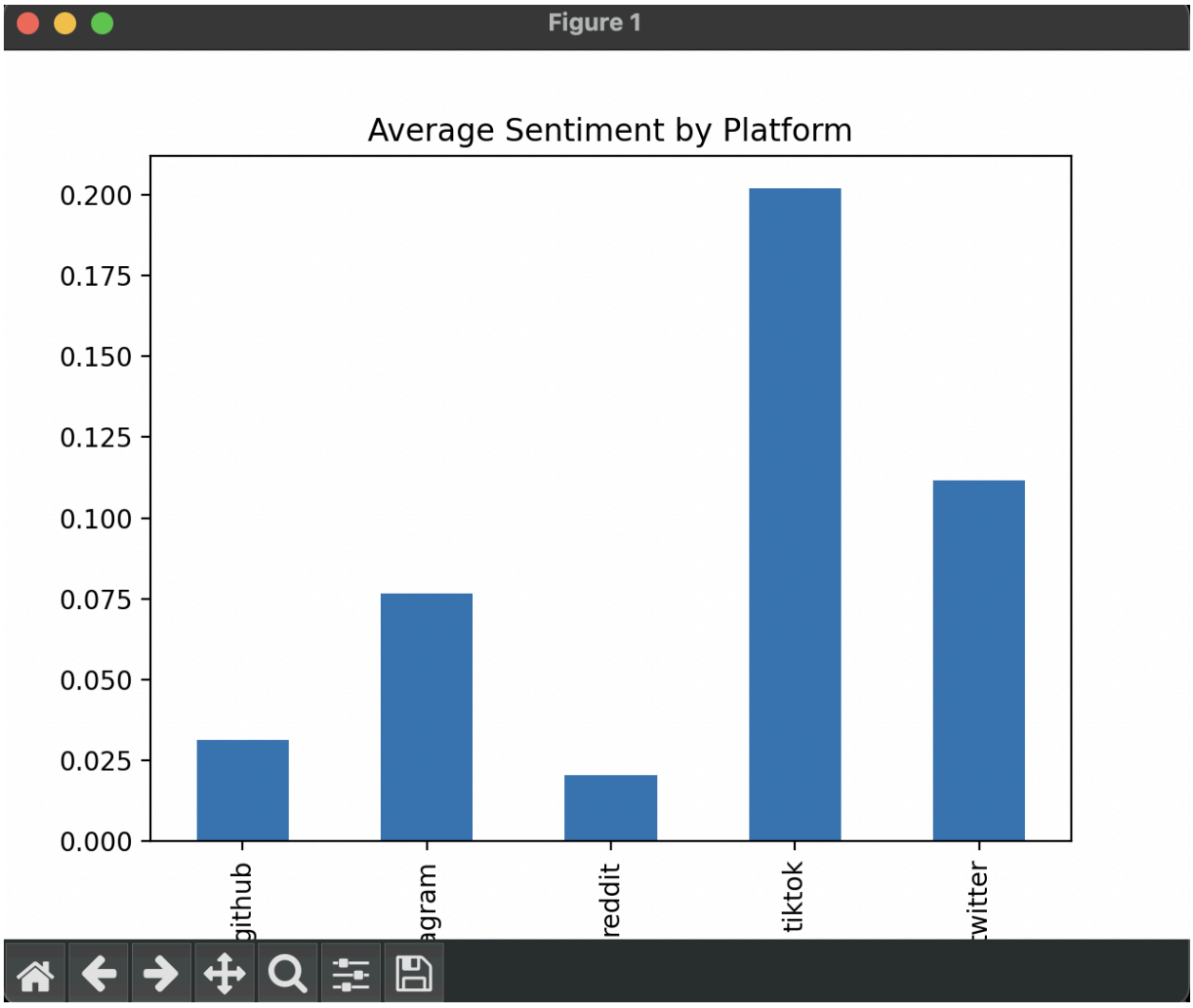
The following table showcases a sample of the records stored in the osint.db database, demonstrating the standardized format and the added sentiment score.

platform	user	timestamp	text	url	sentiment
twitter	techReviewer	2025-10-03 18:45:10	The new AI model is incredibly powerful and impressive	https://twitter.com/...	0.85
reddit	DataIsCool	1664821510.0	I built a tool to analyze market trends It works great	https://reddit.com/...	0.80
github	dev_ops_guru	2025-10-01 10:20:00	Leaked credentials found in public repository	https://github.com/...	-0.60
facebook	cnn	2025-10-03T12:00:00+0000	Breaking news report on the recent global summit	https://facebook.com/...	0.00
tiktok	cybersecurity_tips	1664811000	Never reuse your passwords	https://tiktok.com/...	-0.40

			Here is why its a huge risk		

3.2. Visualizations

As part of the analysis, a bar chart was generated to compare the average sentiment polarity across different social media platforms. This visualization provides a high-level overview of the general tone of conversation on each platform for the searched keywords.



A bar chart titled **Average Sentiment by Platform** would be displayed here. The Y-axis

would represent the average sentiment score (from -1 to 1), and the X-axis would list the platforms (Twitter, Reddit, etc.). This chart would visually represent which platforms host more positive, negative, or neutral discussions on the queried topics.

4. Challenges

Several challenges were encountered during the development and execution of the OSINT pipeline.

- **API Restrictions and Authentication:** Each platform has a unique process for developer registration and API key issuance. LinkedIn, in particular, has very strict API restrictions, necessitating the use of a library that simulates a user session, which is less stable. Managing the various keys, tokens, and secrets required a secure and organized approach using a
  .env file.

- **Rate Limiting:** Most APIs impose rate limits on the number of requests allowed within a specific time frame. This required careful management of API calls to avoid being temporarily blocked.
- **Inconsistent Data Formats:** Data returned from each platform's API varied significantly in structure. A major task was to parse these different formats and map them to a single, unified schema (
  platform, user, timestamp, text, url) to allow for consistent processing and storage.

- **Scraping Fragility:** For platforms without an official public API, such as Quora, web scraping was used as a fallback. This method is fragile and can break whenever the website's front-end structure is updated.

- **Unofficial and Experimental APIs:** The integration for TikTok relied on an unofficial API , and the Snapchat collector was marked as experimental. These integrations are inherently less reliable and may lack proper documentation or support.

5. Conclusion and Future Improvements

This project successfully demonstrated the feasibility of building an automated, multi-platform OSINT aggregation pipeline. The final system is capable of continuously collecting public data, cleaning and enriching it with sentiment analysis, and storing it for intelligence purposes. The modular design allows for easy expansion and maintenance.

While the current pipeline is fully functional, several enhancements could be implemented in the future.

- **Advanced Data Analysis:** The installed nltk and scikit-learn libraries can be leveraged for more advanced Natural Language Processing (NLP) tasks, such as Named Entity Recognition (NER) to extract people, organizations, and locations; Topic Modeling to identify key themes in discussions; and trend detection to spot emerging narratives.

- **Enhanced Scalability:** The current implementation uses SQLite, which is suitable for small-to-medium scale collection. For larger operations, migrating the database to a more robust system like PostgreSQL or a NoSQL database like MongoDB would improve performance and scalability.

- **Interactive Dashboard:** The matplotlib-based visualization could be replaced with an interactive web-based dashboard using frameworks like Dash or Streamlit. This would allow users to filter data in real-time and explore the collected intelligence dynamically.

- **Improved Error Handling:** More sophisticated error handling and logging can be integrated into each collector to make the pipeline more resilient to network issues, API downtimes, or changes in API responses.