Structured Credit News ETL Dashboard

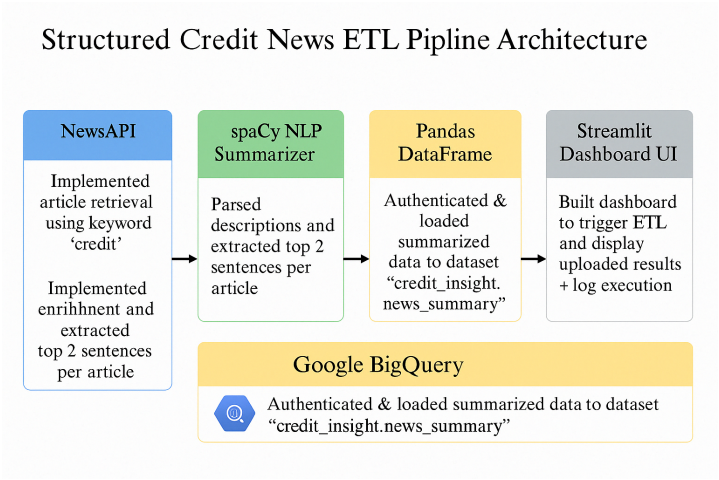
The Structured Credit News ETL Dashboard project focuses on automating the extraction and summarization of structured credit news and storing this data in the cloud. Structured credit encompasses financial products such as CDOs, CMBS and additional asset-backed securities. Due to market volatility and the necessity for timely financial news we aimed to develop a system that delivers summarized structured credit news information through modern data engineering technologies. Our intended system consists of a pipeline that retrieves news articles from NewsAPI for summarization with spaCy and stores the output in Google BigQuery before displaying it on a Streamlit dashboard. The primary consumers of this data would be structured credit analysts and financial professionals seeking timely, summarized insights to support investment decisions. The end product delivers almost real-time analysis of significant news events without requiring extensive human oversight.We originally explored Airflow for orchestration but faced technical friction due to Docker configuration and environmental issues. For MVP, we adopted a manual execution approach via Streamlit to ensure the core pipeline was robust and stable.

Technologies Used

Source System	NewsAPI, FredAPI (Federal Reserve Economic Data) for Macro data
Retrieval method	API-based JSON fetching
Velocity	On-demand or daily
Variety	Textual unstructured data
Volume	Up to 100 articles per execution
Destination	Google BigQuery table *(credit_insight.news_summary)*

ETL architecture

Extract	Pull articles matching the keyword "credit" from NewsAPI
Transform	Use spaCy to extract key sentence-level summaries
Load	Store the results as structured rows in BigQuery
Interface	Streamlit dashboard to trigger ETL and view results



Project Work Review

I started my search for financial news APIs and selected NewsAPI because of its extensive documentation and straightforward integration process. I tested multiple NLP libraries and ultimately selected spaCy because of its streamlined and effective summarization functionalities. The most difficult aspect was environment stability. Multiple dependency and configuration problems arose when installing and running Airflow locally because the necessary containers were both large and complex. The handling of BigQuery credentials required secure management across both local and cloud execution environments. The news content proved rich but frequently featured absent or mismatched *descriptionfields*. We developed strong fallback logic to handle absent fields and convert formats. Maintaining consistent *datetime* format handling proved essential for achieving BigQuery compatibility. Our initial plan was to use Airflow for setting up automatic daily execution schedules. System time constraints and Docker integration failures led us to switch to a Streamlit-based manual triggering solution. A feature to record ETL execution history in CSV format has been incorporated so that users or future scripts can monitor past runs. We deviated from our original plan through the simplification of our scheduling and monitoring setup to prioritize effective data movement along with interface usability. The primary pipeline functions extract data, summarize results, and store them functioned correctly without deviations from our initial design.

Project Retrospective

A modular ETL framework facilitated swift testing and prototyping in Streamlit, BigQuery delivered scalable and cloud-native storage capabilities. The spaCy summarization feature produced clear sentence-level insights from financial news data with strong performance. We initially proposed using Airflow for scheduling our tasks but encountered configuration and Docker-related challenges that compelled us to switch to manual process execution using Streamlit. The change in strategy preserved a manageable project size and confirmed that essential functions were properly developed and evaluated. The most challenging aspects of the project included stabilizing environments and managing authentication flows between local and cloud execution. The team implemented strong fallback solutions and performed multiple adjustments to manage missing fields and inconsistent datetime formats. A managed orchestration tool like Cloud Composer or Prefect Cloud could have been beneficial from the start to lessen development friction and enhance automation. During the execution of our on-demand pipeline I realized that streaming solutions such as Kafka or Pub/Sub would become beneficial for future iterations when real-time monitoring becomes necessary. Should I enroll in an advanced data engineering class I would prioritize CI/CD pipelines along with Kubernetes-based deployment and distributed systems designed for real-time processing. In future developments, I would seek access to a cloud-based orchestration platform and a skilled cloud DevOps team along with diverse news source integrations including Bloomberg and LexisNexis to enhance data quality and minimize vendor dependence. The project demonstrated the need for thorough scope management alongside infrastructure design and resilient system development that manages incomplete or inconsistent data effectively. The project expanded my knowledge about designing systems that go beyond functional pipelines to achieve sustainability and scalability.