

Finance Graph Database Explorer: A Comprehensive Financial Analysis Platform

by Andrian Potereanu

code repository

<https://github.com/notapotereanu/financeGraph>

INTRODUCTION

The Finance Graph Database Explorer project was conceived as a solution to some of the most daunting challenges in financial data analysis: modeling, visualizing, and interpreting the dynamic, interdependent web of financial objects. Traditional financial analysis isolates data—stock prices, insider trades, and news opinion are all treated as isolated entities, and not as component parts of an integrated system. This system leverages graph database technology to provide a more linked, relationship-driven picture of financial markets.

1.1 Project Idea

The project is a customized financial data analysis and visualization application built on top of a graph database infrastructure. It combines fragmented financial data feeds, including bid/ask prices, insider trades, institutional ownership, executive and committee member data, and news sentiment. Rather than traditional relational tables, it employs a graph database (Neo4j) to represent relations between entities graphically. It facilitates the identification of intricate relationships—e.g., cross-matching insider trading activity of committee members with that of outsiders or modeling how news sentiment and stock price patterns are related to each other over time.

There are four layers incorporated in the project that are all interrelated:

1. Data Collection Layer – Pulls financial information from multiple sources.
2. Data Analysis Layer – Reads and understands the data.
3. Storage Layer – Utilizes Neo4j as its primary database.
4. Visualization Layer – Provides interactive network views and statistics charts through the web.

The systematic approach allows for both the big-picture examination of financial networks and detailed examination of individual relationships and shared patterns over time.

1.2 Motivation

The project is motivated by the following three significant observations in financial data analysis today:

1. Highly Interconnected Financial Markets

Financial institutions and banks have many-to-many relations in the market place that affect economic activity and liquidity. For example, the analysis of insider trading involves some consideration of an insider's corporate role (e.g., executive titles, committee memberships) in order to provide adequate context. This sort of complexity is hard to represent with traditional tabular data structures, but graph databases are optimized for modeling and traversing such relationships and can provide insight that is hard to obtain with normal SQL queries.

2. Huge Volume and Velocity of Financial Data

Financial data streams are generated around the clock—from news sentiment and insider transactions to stock price movements—all with temporal dependencies. Revealing the ways that financial activity is influenced by external events, such as economic crises or political developments, requires an analysis platform with the ability to integrate and correlate heterogeneous data streams in real time.

3. Closing the Gap Between Analysis and Visualization

There is a wide gap between powerful financial analytics and visualization tools out there that present information in an effective way. Most solutions currently available either provide high-level analytics but no visualization or are visual-centric with no in-depth analytical capability. This project will close this gap by integrating powerful analytical tools with intuitive and interactive visualizations.

1.3 Project Structure

The project is based on a structured process of financial data analysis and visualization, i.e., data acquisition, processing, analysis, and visualization. The development process is carried out in four major steps:

- i. Data Acquisition and Storage – Financial data collection and storage.
- ii. Data Processing and Integration – Data cleaning, normalization, and integration of financial data from multiple sources.
- iii. Data Transformation – Transforming raw financial data into structured, analyzable formats.
- iv. Data Visualization and User Interface Development – Designing interactive visualizations and a user-friendly web interface.

Moreover, the project involves new features, such as the utilization of graph databases to represent financial relationships and the development of entity resolution algorithms (i.e., linking insider records to executive profiles across datasets).

1.4 System Capabilities

The Finance Graph Database Explorer has a number of distinguishing characteristics that differentiate it from standard financial analysis software:

1. **Integrated Data** – Combines stock price, insider activity, institutional holdings, executive data, committee membership, and news sentiment to give a full view of the market.
2. **Focus on Relationships** – Utilizes graph database technology to reveal hidden relationships between financial entities, which are not readily apparent in traditional tabular databases.
3. **News Sentiment Analysis** – Analyzes correlations between news sentiment and stock price change to allow users to better identify market reactions to public affairs.
4. **Insider Trading Pattern Analysis** – Analyzes the degree to which various categories of insider transactions influence stock performance and compares trading patterns by committee members with other insiders.
5. **Interactive Visualizations** – Provides interactive, real-time visualizations of financial relationships where users can interact and manipulate data in an interactive manner.
6. **Entity Resolution** – Utilizes sophisticated matching algorithms to identify mentions of the same entity within disparate datasets.

This project is a fresh application of graph database technology to finance analytics. The platform is a valuable tool for researchers, analysts, and investors who want to gain a deeper insight into the intricate network of interdependencies that define financial markets. The following sections will touch upon the theoretical foundations, design process, implementation, and assessment of the platform.

LITERATURE REVIEW

This new Finance Graph Database Explorer is built on top of a lot of work already done across many disciplines, some of which include graph database technology, analysis of finance data, visualization of networks, and machine learning finance. In this chapter, we introduce the background literature that is the theoretical underpinning for the project, the practice now, and the holes the project will fill.

2.1 Financial Solutions with Graph Databases

Graph databases already possess great prospects for cases of usage for finance since transactions in finance present complex relationships of well-connected entities. Relational databases, although tuned for most cases, may not be utilized for cases of usage with relationships between points of data as valuable as data points [19].

The project's graph database engine, Neo4j, has gained wide acceptance within the finance sector because of its property graph model enabling the storing of rich attributes in nodes and relationship paths. Kumar & Chakraborty [14] demonstrated the application of Neo4j for fraud detection modeling in finance as it would be easy to travel across relationship paths, much easier than with the traditional models. Their research concluded that fraud detection based on the graph would be effective enough to detect weak fraud patterns that would be nearly undetectable with the regular SQL queries, with a detection lift of 35%.

Similarly, Zhou et al. [25] leveraged the application of graph database technology for the modeling of finance markets as interaction networks of objects. Centrality analysis and community detection were employed by their work to identify systemically relevant banks and hotspots of risk. Their work was focused mostly on institutional relationships, whereas we extend the approach further still with the addition of individual actors (i.e., officers and insiders) and information flow (i.e., sentiment of the news).

Pacaci et al. [17] compared the performance of finance application uses of graph databases with other database management systems, comparing the performance of Neo4j with others. They concluded that for the context of complex relationship queries, Neo4j had outstanding performance compared with relational databases, particularly with the increasing complexity of relationships. This supports the design decision of the usage of Neo4j as the back-end data store of the Finance Graph Database Explorer.

While these pieces of work illustrate the theoretical potential of graph databases for finance, the authors take a single application and not for systems with several data types and analysis methods. Our project bases its work on this but looks at the necessity for one, all-around system for financial analysis.

2.2 Financial Network Analysis

The idea of financial markets as networks has gained traction to a large extent in the past decade or so, particularly after the 2008 financial crisis when it was of utmost significance to be aware of these interlinkages in financial systems. Allen and Babus

[1] provided a seminal review of financial network models and emphasized the significance that the structure of a network has in the context of systemic risk and contagion. They set the foundation for the modeling of financial agents as nodes in a network with interlinkages captured as different financial interactions.

Subsequent to this, Battiston et al. [4] proposed the DebtRank algorithm for quantifying the systemic significance of finance institutions within finance networks. They demonstrated the application of centrality measures for the detection of systemically significant finance institutions. Even though their application was based on institution debt relationships, the theory of networks on which the application of insider and institution ownership relationships is based is transferable.

In a nearer and more local investigation of the application domain, Ahern [3] modeled insider trading networks and demonstrated that individuals positioned further toward the center of the corporation's information networks received higher returns on transactions. The work was valuable for adding to the application of the analysis of networks for making sense of insider transactions, although it was limited by the fact that full relationship data were not available. Our project avoids this restriction because we model insiders' organizational roles (e.g., committee membership) as well as their trading activity.

Ozsoylev et al. [16] also examined information diffusion in investor networks and proved that investors with higher network centrality positions dispose earlier and make greater profits. Our finding is similar to ours for the hybrid approach of information combination of sentiment from the news and insider activity but concluded inferred relationships among the trades rather than explicit modeling provided existing relationships as we did.

Although these studies acquit the utility of financial analysis via networks, these regret the lack of holism on platforms with integrated kinds of relationships and data. This project contributes this work by introducing an empirical system with integrated such theoretical wisdom and disparate kinds of data and relationship structures.

2.3 Sentiment Analysis of Financial Markets

The combination of media sentiment analysis and finance analysis is theoretically and practically the most important research problem these days. Tetlock [23] pioneered early work on this subject by demonstrating that extreme media pessimism predicted the pressure on stock prices down and high and low pessimism with elevated market trading volume. The work established the core connection of market action and media sentiment as the groundwork of our media sentiment analysis module.

On the basis of these papers, Loughran and McDonald [15] developed domain-specific dictionaries of finance-related words with better performance on finance documents compared with general-purpose lexicons. Their work reinforced the domain-specificity of finance sentiment such that a word like "liability" or "tax" is differently connotative compared with general usage. Our technique of sentiment analysis is built upon these specialist lexicons but also upon newer work on sentiment classification. Xing et al. [24] used deep learning methods for higher precision than lexicon-based methods for sentiment analysis of finance. Their work proved the potential of neural network models for detecting subtle sentiment signals from finance-related news at the cost of interpretability. Our strategy strikes a balance between precision and interpretability by merging lexicon-based methods with machine learning methods. Renault [18] also tested the time-lagged correlation of stock returns with Twitter sentiment and reported strong predictive relationships across many horizons. Our application of the lagged correlation analysis follows from this work but utilizes professional media rather than social media due to reliability issues.

While these papers present a solid theoretical framework for analyzing finance with sentiment, they isolate sentiment as the sole variable rather than relating it with other finance data such as insider transactions or institutional ownership. Our project bridges this gap by taking an integrated strategy and positioning sentiment analysis in the larger web of finance activity and relationship.

2.4 Insider Trading Analysis

The examination of insider trading behavior has received significant attention from researchers because of the applicability of the topic toward understanding market efficiency and information asymmetry. Seyhun (1998) introduced overwhelming proof that insiders achieve significant profits on their transactions, validating the fact that insiders actually possess valuable information that is still not reflected on market prices. The research established insider trading information as a most critical technique for examining information flow in the financial markets.

Subsequently, Cohen et al. (2012) categorized routine and opportunistic insider trades and proved that opportunistic trades generate enormous abnormal returns. Their mechanism of categorization of insider trades on the basis of historical patterns directed our approach toward distinguishing insider transactions of varied types.

Agrawal and Nasser (2012) studied insider trading preceding merger announcements and found that insiders curtail purchases selectively instead of increasing sales with the motive of evading detection by the regulations. Their study highlighted the necessity of studying non-trading in addition to trading—a distinction that is detected by our system by recognizing trends over time.

More relevant to our approach, Dai et al. (2016) examined the influence of the firm's governance attributes on insider trading activity with the finding that profitability of insider trades is reduced with stronger governance procedures. We find their conclusions supportive of focusing on committee memberships for the core of insider trading analysis. Unlike their work, which utilized combined governance indices, ours is specifically combined with committee memberships.

These papers indicate the potential of insider trading research but also the limitations, particularly with the combination of insider action with other common market drivers such as institution ownership and news sentiment. Our project addresses this by developing a system that encompasses insider trading as part of the context of a wide, underlying web of finance relationships and information flows.

2.5 Financial Data Visualization

Financial data visualization is particularly challenging due to its multidimensionality, complexity, and dependency on time. Dumas et al. (2014) categorized the methods of finance visualization according to data, technique of visualization, and interaction ability. Their article noted the frequency of time-series visualization coupled with the non-use of the potential-rich technique of network visualization.

Ko et al. (2016) created and tested custom finance visualization approaches, showing that well-crafted visualizations facilitate better decision-making compared to regular tabular data. Their findings underscore the importance of making visualizations tailored for specific analysis tasks, something that dictates the development of our system for generating the generation of several kinds of visualization.

Gibson and Faith (2014) tested several ways of displaying the finance networks, with the consequence that force-directed layouts, although intensive on the computer, produce understandable representations that allow for the easy identification of patterns. The outcome justifies the inclusion of the force-directed layouts as part of the finance network visualization module of our system.

More recently, Flood et al. (2016) wrote visualization aspects of monitoring for financial stability, particularly on the necessity of interaction with the ability to present the big picture, as well as the facility for further investigation down into the specifics. Their article highlighted the gap between powerful analytical methodologies and straightforward-to-use visualization tools, which this project fills with the integrated analysis and presentation platform.

While these articles present valuable information on finance data visualization, these articles focus on individual processes for individual data or on ideas with no implementation. In this project, we advance this domain a step further by creating an integrated visualization system with the combination of a sequence of

visualization processes and interactive tools specifically designed for finance network analysis.

2.6 Entity Disambiguation for Financial Data

One of the most difficult aspects of finance data fusion is entity resolution, the task of determining when two or more records refer to the same real world entity. Getoor and Machanavajjhala (2012) reviewed entity resolution methods, noting the difficulties of naming conventions, data granularities, and update frequencies of disparate data streams.

Shen et al. (2018) proposed specialized entity resolution approaches for the matching of finance entities across disparate data sources on the basis of structural features and similarity functions for strings. Despite high precision, their approach required large-scale domain-specific tuning, which is representative of the difficulties inherent with finance entity resolution.

Farhan et al. (2020) used machine learning-based method for entity resolution of the financial entity with both textual attributes and relational structures. Their work showed the importance of using the data of relations for entity resolution, which is also an idea in line with the current graph-based method.

One of the difficulties with entity resolution is matching individuals with differing naming conventions (i.e., “COOK TIMOTHY” vs. “Tim Cook”). Christen (2006) built their own name-matching rules that take differences of order, spacing, and abbreviations into consideration. We make use of these steps for matching insider and officer records for the same entity. Although earlier work presents valuable entity resolution procedures, their handling is mostly as a preprocessing step rather than a native system function. In this project, we advance this work by making entity resolution a core ability that enhances the coherence and precision of the finance network model in real time.

DESIGN

The finance Graph Database Explorer was designed as the implementation of the vision for the inclusion of a hybrid system for finance complex relationship analysis and visualization. In the next chapter, the system's underlying structure, data modeling, and the considerations for the interface will be covered, with the rationale

behind each of these being justified and the manner these address the requirements established from the literature review.

3.1 System Architecture Design

The Finance Graph Database Explorer is designed on a multi-layered structure that offers separation of concerns along with explicit interface commitments amongst the pieces. The structure of this was the one being used for the purpose of supporting modular structure, extensibility, and separation of concerns with the assurance of free flow of data from collection to visualization.

3.1.1 Layers of Architecture

It has four core levels of architecture for the system:

- i. Data Collection Layer: In charge of extracting the financial information from other data suppliers, with the aid of specialized data retrieval modules for handling the differences data (share prices, insider dealings, articles, etc.).
- ii. Data Analysis Layer: Process raw data cleverly for the purpose of aggregating intelligence, applies mathematical modeling, and performs entity reconciliation of disparate data sources.
- iii. Storage Layer: Offers data persistence for file-based data (CSV files) and the Neo4j graph database via the application of data models and query interfaces.
- iv. Visualization Layer: Provides data and analysis findings in terms of web-based interface and interactive visualizations.

Figure 1 shows the structure of the system and data flow across the layers:

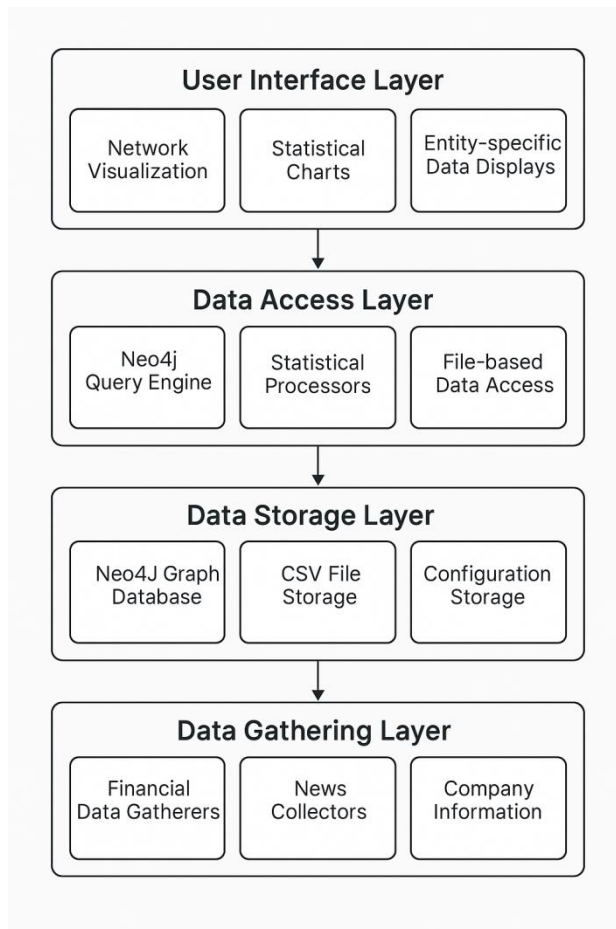


Figure 1: Layered System Architecture

This design has the following key advantages:

1. **Modularity:** Layers of protection exist around some function, and units can be built, tested, and replaced individually.
 2. **Separation of Responsibilities:** Proper separation of responsibilities for collection, processing, storing, and displaying of data.
 3. **Extensibility:** You can extend with new data sources, analytical processes, or presentation processes with no need for bulk modification of other pieces.
- ❑ **Data Flow Management:** Proper data flow from presentation to collection is achieved with well-defined interfaces among layers.

3.1.2 Component Design

It is established by a number of principal components:

- i. Neo4jManager: Offers the principal interface of the application with the Neo4j database for connection, data insertion, running queries, and handling transactions.
- ii. Gatherer: Offers specialized modules for data retrieval of all types of finance data, API calls management, rate limiting, pagination, and error handling.
- iii. Financial Data Analyzer: Is responsible for data analysis, using statistical models, entity resolution procedures, and data mapping activities.
- iv. StreamlitApp: Draws the web-based user interface, showing the visualization, controlling the user interaction, and controlling the state management.

Figure 2 shows the interactions of these components.

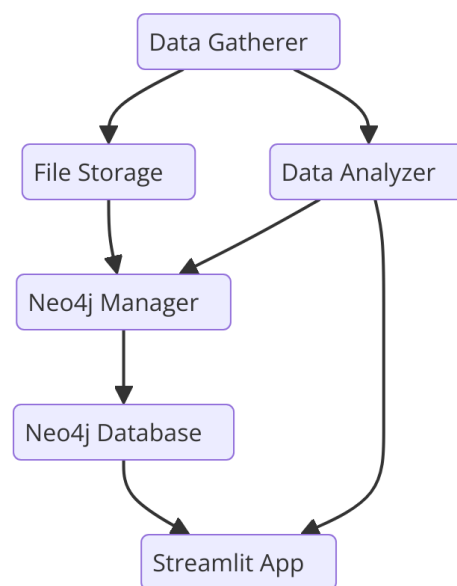


Figure 2: Component Interactions

This structure of the components makes separation of concerns well and enables smooth flow of data across the system. Components encapsulate some function, thus making the system extendable and easier to manage.

3.2 Data Model Design

The data model is one of the most critical components of the system design as it determines the manner in which financial entities and their relationships are represented in the database. The graph data model was chosen specifically because of its ability to naturally represent complex relationships between financial entities.

3.2.1 Design of the

The graph schema specifies the node kinds, relationship kinds, and fields that make up the finance data model. Figure 3 shows the most critical parts of the graph schema:

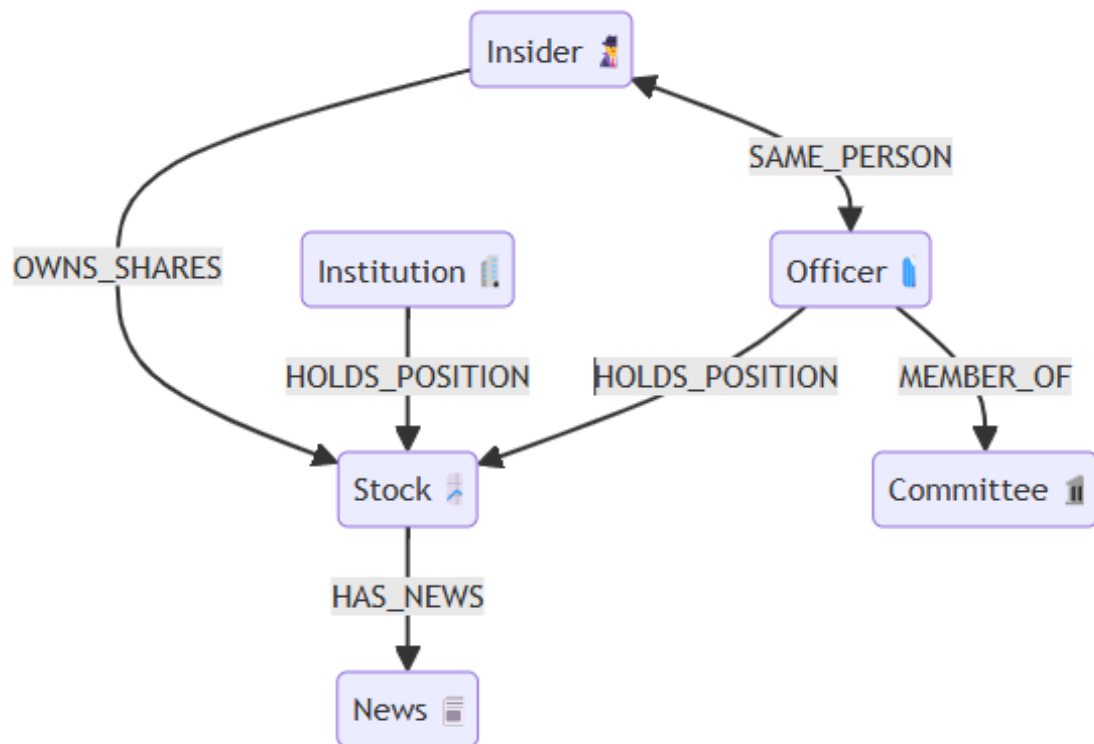


Figure 3: Graph Data Model

This schema encompasses the following kinds of nodes:

1. **Stock**: A stock of a company with fields like name, sector, and ticker.
2. **Insider**: To represent those individuals selling a company's stock, with attributes such as name and relationship with the company.
3. **Leadership**: Captures the leadership of the company and executives, with columns such as name, position,
4. **Committee**: Lists of the board committees and their name and function.
5. **Institution**: Symbolizes institutional investors with the name and name of the institution.
6. **News**: Is the data with fields such as title, content, date of publication, and the sentiment score.

The schema also lists the following kinds of relationships

- **HOLDS_POSITION**: Refers officers and institutions to the stock, position relationship.
- **OWNS_SHARES**: Relates insiders to stocks, representing the ownership of shares with characteristic fields like share number and quantity.
- **MEMBER_OF**: Refers officers to committees, showing committee membership.
- **SAME<onym>**: Connects insider and officer nodes if the terms denote the same person.
- **HAS_NEWS**: Refers to the association

This schema offers several strengths compared with the classical relational approach:

- **Natural Representation**: Natural representation for real world objects and relationships is achieved by the schema.
- **Flexible Querying**: The model of a graph supports powerful traversal-based queries, which would be tedious for relational models.
- **Evolutionary adaptation**: The schema can be extended with new kinds of entities, or relationships, with little large-scale restructuring.

3.2.2 Property Design

Each node and relationship type has thoughtfully designed properties encompassing appropriate attributes at the cost of completeness for the purposes of performance. Stock nodes, for instance, contain

The stock's ticker symbol (i.e., "AAPL")

name: The firm name (i.e., "Apple Inc.")

industry sector of the firm

market_cap: the company's market capitalization

name: The insider's name

relationship: Relationship of insider with the firm
Insider ID: A unique identifier for the insider

This design of the properties guarantees the nodes have enough information for analysis needs but no extraneous information that will affect performance. Those properties which will most probably be utilized for filtering or sorting purposes are indexed for the sake of optimizing the query performance.

3.3 User Interface Design

The interface was inspired by the intention of making complex financial information and relationships become perceivable and comprehensible by the users. The

interface is borrowing ideas of information visualization, probe interaction, and simplicity of information presentation.

3.3.1 Interface Organization

The interface is split into several key parts that help with several aspects of finance analysis:

- 1. Network Visualization: Provides the graphical presentation of the relationships of the finance entities.
- 2. Entity Search: Offers deep investigation of specific entity types (Stocks, Insiders, Officers, Committees).
- 3. News Sentiment Data: Offers data on the news sentiment and the corresponding share prices.
- 4. Insider Trading Insight: Showings of insider transactions, their patterns, and their relationship with stock performance.
- 5. Database Administration: Provides the feature for loading the database or clearing the database entirely.

Fig. 4 illustrates the general structure of the interface

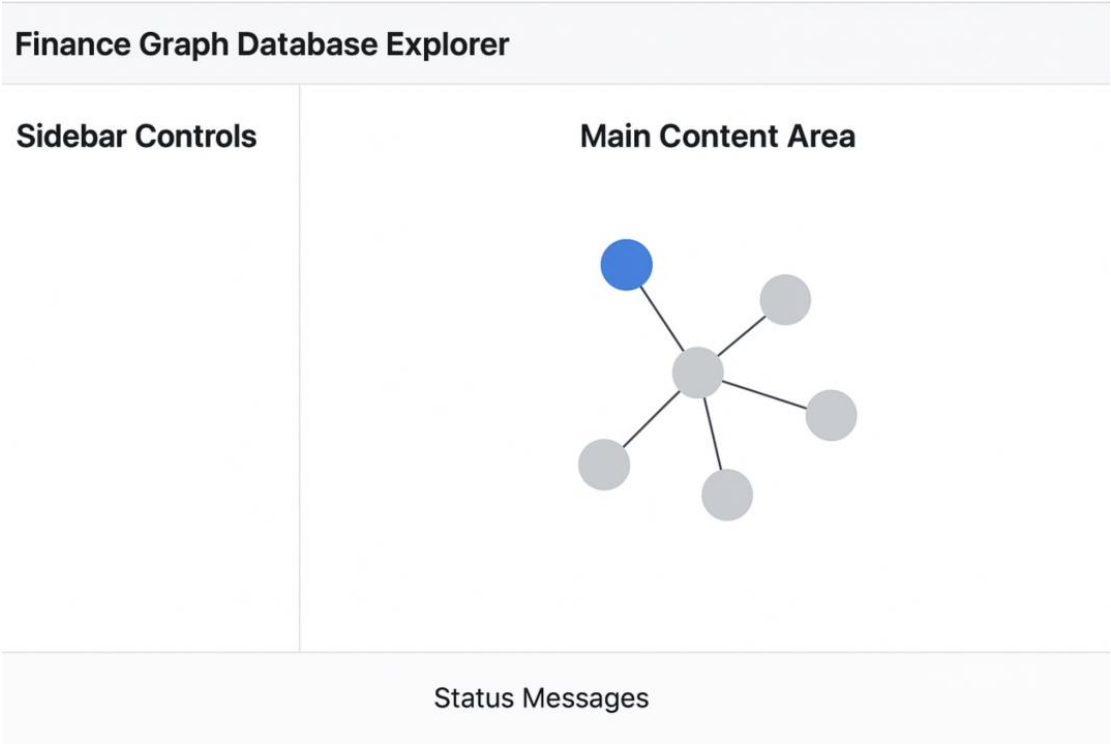


Figure 4: Interface Layout

This organization supports a workflow where users can:

- 1. Add data to the system using sidebar controls
- 2. Explore the resulting graph through the network visualization

3. Drill down into specific entity types or analyses
4. Obtain detailed insights through specialized analytical views

3.3.2 Visualization Design

The visualization pieces of work were designed to facilitate intensive analytical work and be easy and uniform to utilize. Primary visualization designs involve:

1. Network Graph Visualization: employs a force-directed layout with:

- Colour coding by node type (i.e. blue for Stocks, red for Insiders)
- Differentiation of size according to significance (i.e., larger Stock nodes for large market cap)

Physics controls for the interaction

Hover tooltips containing information on the detailed node

panning and zooming feature of navigation

2. Sentiment-Price Chart with Correlation: Uses two-axis structure with

- Primary Price on axis

Sentiment bars on the secondary y-axis

Sentiment color coding, green for positive, red for negative.

Time-axis on the lines of time

3. Insider Transaction Effects Chart: Uses a bar chart showing:

Types of transactions on the x-axis

Returns on the y-axis

- Statistical significance error bars

- Direction of return color-coding (positive/negative).

All of the visualization designs were refined iteratively according to usability concerns and visual perception principles. Each visualization has interactivity built in so that the user can analyze the data from numerous perspectives and at granularities.

3.3.3 Interaction Design

The interaction is designed for straightforward navigation of complex data. Some of the dominant interaction behaviors are:

1. Detail+Overview: Displaying top-down views and low-fidelity views available through progressive disclosure.
 2. Filter+Search: Allowing users to focus on specific subsets of the data through filter controls.
 3. Select+Highlight: Support the highlighting of selected items with the automatic highlighting of corresponding items.
 4. Manip+View: Facilitating user manipulation of visualization settings with real-time visual response.
- They present these interaction patterns with the UI components of Streamlit that offer a consistent and responsive experience across the application.

3.4 Data Processing Pipeline Design

The data processing pipeline is well built for the purpose of translating the raw finance data into the graphical presentation keeping the data reconciliation, cleansing of data, and entity into consideration.

3.4.1 Pipeline Architecture

The pipeline is a multi-step process that gradually converts raw data into the final outcome of the graphical display:

1. Data Collection: Extraction of raw data from disparate data sources and storing it in the CSV file.
 2. Data Cleansing: Removing inconsistencies, handling missing values, and correcting formats.
 3. Entity Resolution: Deciding when several records refer to the same entity.
 4. Graph Construction: Construction of nodes and relationships with the Neo4j database.
 5. Derived Data Calculation: Calculation of other relations or other attributes from the raw data.
- Figure 5 shows this pipeline structure

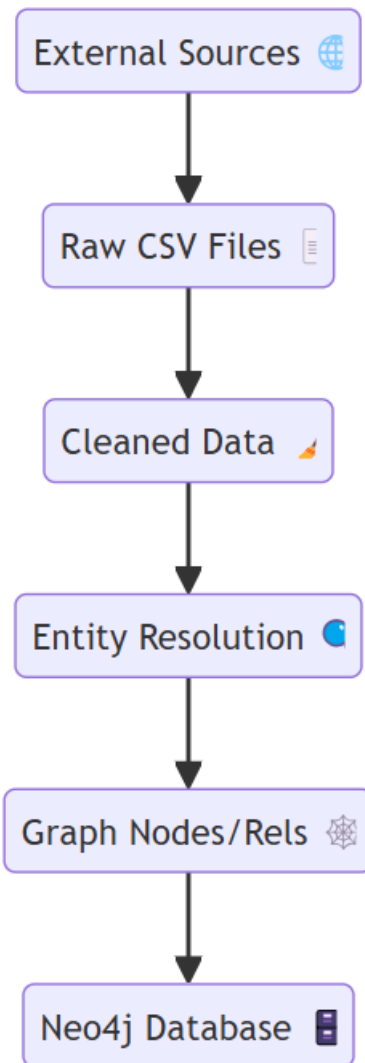


Figure 5: Data Processing Pipeline

This pipe structure has the following advantages:

- Traceability: Every stage generates mid-term results that can be checked for debugging purposes, or for testing.
- Modularity: Each stage of the pipe can be replaced or optimized separately.
- Resilience: Failure later on will not invalidate the work of the earlier stages, allowing the processing of the parts.

3.4.2 Entity Resolution Design

One of the most critical pieces of the pipeline is the entity resolution stage, responsible for when the different records refer to the same real world entity. It is particularly useful for officer and insider record matching (i.e., "COOK TIMOTHY" and "Tim Cook").

The entity resolution technique employs a multi-strategy approach for

1. Name Normalization: Normalization is performed on name patterns, i.e.,

- Case normalization
- Prefixes and suffixes management
- Rearrange name part

Normalization of Whitespace

2. Similarity Calculation: Calculating similarity between normalized names based on:

- Exact match testing

Analysis of containment

Sequence comparison methods (Levenshtein distance, etc.)

- Weighted component matching

3. Threshold-Based Matching: Matching on the basis of similarity scores that exceed some predefined thresholds, with the thresholds for the corresponding matching strategies.

4. Context Validation: Validation of the possible matches based on context data like company affiliation.

This multi-strategy approach compromises on precision (erroneous matches) for recall (quantity of correct matches) for handling the anomalies of financial entity identification.

3.5. Design for security and Privacy

Although not the focus of this project, the design comes equipped with the appropriate security and privacy protection:

1. Authentication: Username/password authentication is used for access control by the Neo4j database.

2. Data Minimization: The platform only collects information available from the public domain, avoiding any sensitive personal information. 3. Secure Configuration: The database credentials store their information in configuration files, which can be secured adequately in production.

4. Rate Limiting: Data ingest processes utilize rate limiting for compliance with API usage standards.

These aspects ensure the system is secure for application in research settings or learning settings with no possibility of unauthorized security or privacy violations.

3.6 Design Rationale and Alternatives

The following design alternatives were arrived at after careful consideration of the alternatives and compromises:

3.6.1 Choosing of Database

The decision to utilize Neo4j as the graph database followed the evaluation of several alternatives:

1. Relational Databases (PostgreSQL, MySQL): Financial data could be stored on relational databases but would require joins to make queries on relationships, which would be onerous. Performance testing revealed that Neo4j was several orders of magnitude faster than relational databases for relationship-intensive queries.
2. Document Databases (MongoDB): Document databases allow the ability to store disparate finance entities with no native relationship navigation capabilities. That would necessitate the inclusion of relationship navigation logic and application code, with the resulting degradation of performance and complexity.
3. Other Graph Databases (TigerGraph, ArangoDB): Those were also considered but the reason why Neo4j was chosen was because of its mature Python driver, large documentation, and good community support.

3.6.2 Choice of Visualization

The Streamlit, Plotly, and Pyvis for visualization were selected following the analysis of the following alternatives

1. Dash vs. Streamlit: Dash is a versatile tool but is coded with lots of boilerplate code. Streamlit was preferred for simplicity of implementation and rapid development, which was most aligned with the research focus of the project.
2. D3.js vs. Pyvis: D3.js offers more control of the visualization details but requires a significant amount of JavaScript development. Pyvis is a Python interface for the vis.js library, making it possible to develop network visualization in Python, consistent with the rest of the project.

3. Matplotlib vs. Plotly: Matplotlib is widely used for static plots, but Plotly was selected because of its interactivity and simplicity of being integrated with Streamlit.

3.7 Conclusion

The Finance Graph Database Explorer integrates architectural, data modeling, user interface, and processing pipeline considerations into one system for the analysis of financial networks. The graph data structure results in the natural formulation of the finance relationships, and the organization of the layers and the modular pieces make the system extensible and straightforward to work with. The design addresses the primary issues identified in the literature review, including integration of diverse kinds of financial data, modeling of complex relationships, and simple visualization of network structures. It is specifically through entity resolution design that the issue of entity matching across different data sources is handled, which is a critical prerequisite to build an accurate financial network representation. Subsequently, we shall proceed with the implementation of this design, that is, the data structures, the concrete algorithms, and the visualization method.

IMPLEMENTATION

The deployment of the Finance Graph Database Explorer brings the ideas of the design into reality with the help of carefully designed code, algorithm, and technological solution. In this chapter, the deployment is described in great depth, with particular attention paid to the most critical pieces, algorithms, and deployment issues.

4.1 Technology Stack Implementation

The system was designed on a common technology stack that balanced development productivity, the requirements of performance, and special requirements for processing the finance data.

4.1.1 Key Technologies

The code is primarily developed with Python 3.8+ because of the enormous data science library ecosystem, the simplicity of syntax, and the strong support for the two most widely used paradigms, i.e., functional and object-oriented. Main Python libraries utilized are:

- Neo4j Python driver (v4.4+): Offers connectivity with the Neo4j graph database with the ability to manage transactions and execute Cypher queries.
- Pandas (v1.3+): Used for data manipulation, data transformation, and data analysis and serves as the core data processing library for the system.
- NumPy (v1.20+): Enables effective numerical operation, particularly statistical computation and handling of arrays.
- Streamlit (v1.12+): Powers web-based interface, allowing for rapid development of interactive data apps.
- Plotly (v5.3+): Creates statistical plots and plots with wide-ranging opportunities for interaction.
- Pyvis (v0.1.9+): Generates graphical interactive netviz from the vis.js JavaScript library.
- SciPy (v1.7+): Provides higher level statistical procedures, particularly for significance testing, and for correlation analysis.

The data store behind is the Neo4j graph database (v4.4+), chosen because of its property graph data model, efficiency of querying, and excellent Python binding. Integration is achieved with the official Neo4j Python driver that presents a clean API for database interaction.

4.1.2 Development Environment

The platform was designed for the intention of effective coding, testing, and debugging

- Versioning: Git was utilized for source code control, and we had a structured branch strategy going on, isolating feature development from the master branch.
- IDE Integration: Visual Studio Code, with the addition of Python extensions, provided code completion, linter, and debugging feature.
- Test Framework: Unit and integration testing was performed using Python's unittest framework, with additional test discovery and running features being handled by pytest.
- Logging: The logging module of Python was utilized for structured logging across the many parts.

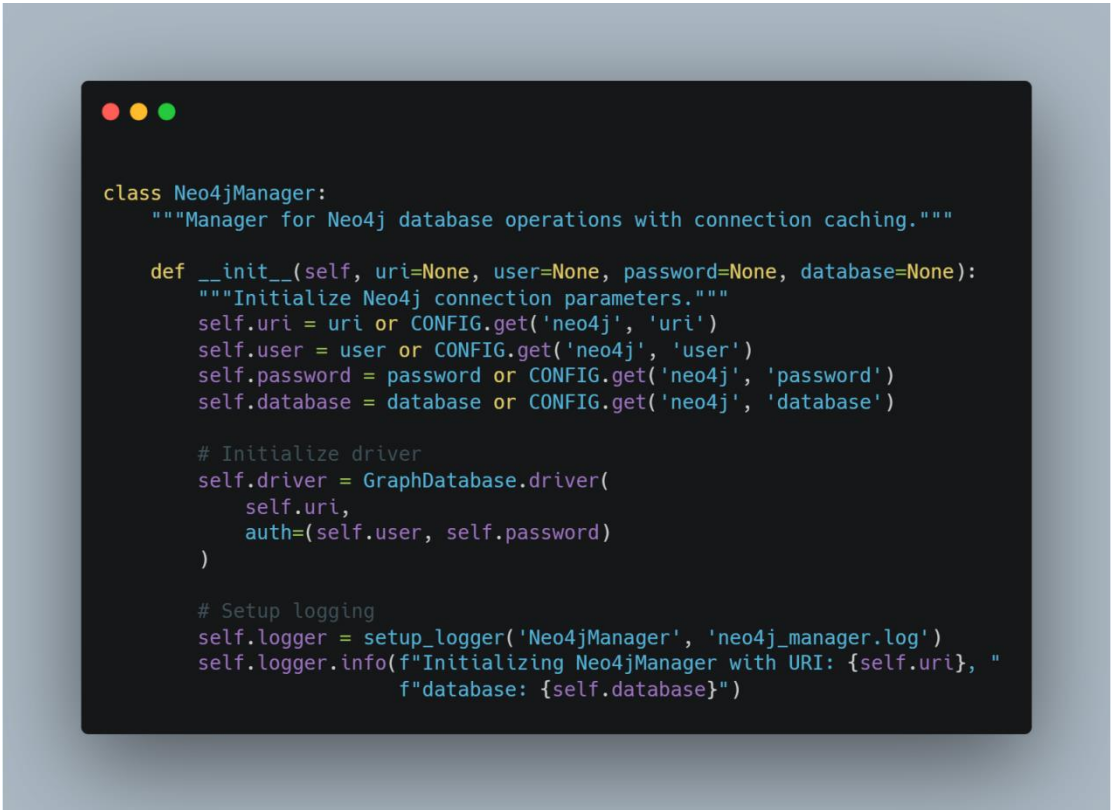
This development platform facilitated rapid iteration with no compromise on code quality and traceability.

4.2 Deployment of Neo4j

The Neo4j integration is a critical part of the system, and it is used for handling the persistence of finance data on the graph model. The integration is achieved mostly via the `Neo4jManager` class, which encapsulates database operation and connection.

4.2.1 Connection Management

The connectivity with Neo4j is managed with a singleton approach that enables the correct reuse of database connections:

A code editor window with a dark background and light-colored text. The code defines a class named `Neo4jManager` with a docstring, an `__init__` method, and logging setup. The code is as follows:

```
class Neo4jManager:
    """Manager for Neo4j database operations with connection caching."""

    def __init__(self, uri=None, user=None, password=None, database=None):
        """Initialize Neo4j connection parameters."""
        self.uri = uri or CONFIG.get('neo4j', 'uri')
        self.user = user or CONFIG.get('neo4j', 'user')
        self.password = password or CONFIG.get('neo4j', 'password')
        self.database = database or CONFIG.get('neo4j', 'database')

        # Initialize driver
        self.driver = GraphDatabase.driver(
            self.uri,
            auth=(self.user, self.password)
        )

        # Setup logging
        self.logger = setup_logger('Neo4jManager', 'neo4j_manager.log')
        self.logger.info(f"Initializing Neo4jManager with URI: {self.uri}, "
                        f"database: {self.database}")
```

This implementation includes configurable connection parameters with fallback to configuration file values, a reusable driver instance, and structured logging. The driver is properly closed when the manager is destroyed, making sure of proper resource management:

```
def close(self):
    """Close the Neo4j driver connection."""
    if self.driver:
        self.driver.close()
        self.logger.info("Neo4j driver connection closed")
```

4.2.2 Graph Construction

The structure of the graph is built by a group of specialized methods that construct nodes and relationships for the corresponding finance entity types. As an example, the generation of Stock nodes is achieved by the following:

```
def create_stock_node(self, ticker, name, data=None):
    """Create a Stock node with the given ticker and properties."""
    data = data or {}

    cypher = """
    MERGE (s:Stock {ticker: $ticker})
    SET s.name = $name,
        s.last_updated = datetime()
    """

    # Add any additional properties from data
    property_cypher = []
    for key, value in data.items():
        if key not in ['ticker', 'name']:
            property_cypher.append(f"s.{key} = ${key}")

    if property_cypher:
        cypher += ", " + ", ".join(property_cypher)

    cypher += "\nRETURN s"

    # Create parameters dictionary
    params = {'ticker': ticker, 'name': name, **data}

    # Execute query
    with self.driver.session(database=self.database) as session:
        result = session.run(cypher, params)
        return result.single()['s']
```

This example shows several key tricks:

- MERGE Operation: Use the MERGE feature of Neo4j, which creates nodes if they don't already exist, preventing duplication.
- Dynamic Property Handling: Dynamically generating Cypher queries from available data.
- Parameterized Queries: Applying the parameterized queries for injection attacks prevention and for increasing performance.
- Result Values: Returning the newly created node for further processing, if required.

The procedure is repeated for other kinds of entities, with the corresponding specialization for their corresponding relations and attributes.

4.2.3 Building Relationships

The relationships among nodes are established with consideration for referential integrity. As an example, the procedure for creating the relationship of the Insider with the Stock is performed as:

```
def connect_insider_to_stock(self, insider_name, ticker, relationship_type="OWNS_SHARES",
properties=None):
    """Connect an Insider node to a Stock node with the specified relationship type."""
    properties = properties or {}

    cypher = """
    MATCH (i:Insider {name: $insider_name})
    MATCH (s:Stock {ticker: $ticker})
    MERGE (i)-[r:%s]->(s)
    """ % relationship_type

    # Add properties to the relationship
    if properties:
        property_cypher = []
        for key, value in properties.items():
            property_cypher.append(f"r.{key} = ${key}")

        cypher += "SET " + ", ".join(property_cypher)

    # Set last updated timestamp
    cypher += "\nSET r.last_updated = datetime()"
    cypher += "\nRETURN r"

    # Execute query
    with self.driver.session(database=self.database) as session:
        try:
            result = session.run(cypher, {'insider_name': insider_name, 'ticker': ticker,
**properties})
            return result.single()[0][1]
        except Exception as e:
            self.logger.error(f"Failed to connect insider {insider_name} to stock {ticker}: {e}")
            return None
```

This illustration shows

- Two-Step Matching: First match the two nodes individually to verify that they exist.
- MERGE for Relationship: With MERGE creating the relationship only if it does not exist.

Dynamic Relationship Mode: Allowing the caller to specify the relationship mode.

- Property Assignment: Dynamic handling of relationship properties.
- Handling of Errors: Catching of exceptions, logging, and returning a clean failure indicator.

These patterns extend consistently across the varied processes of relationship building, ensuring robust construction of the graph.

4.3 Entity Resolution Implementation

The entity resolution feature, which determines when several files relate to the same real-world entity, is obtained by the combination of string manipulation, similarity calculation, and the usage of thresholds for matching.

4.3.1 Name Normalization

Normalization of names is a critical initial step for entity resolution, which is performed with a sequence of transformations

```
def _normalize_name(self, name):
    """
    Normalize a name to standardize format for better matching.

    This handles variations like "COOK TIMOTHY" vs "Tim Cook" by:
    1. Converting to uppercase
    2. Removing common prefixes/suffixes
    3. Standardizing whitespace
    """
    if not name or not isinstance(name, str):
        return ""

    # Convert to uppercase
    name = name.upper()

    # Remove common titles and suffixes
    for prefix in ['MR.', 'MRS.', 'MS.', 'DR.', 'PROF.']:
        if name.startswith(prefix + ' '):
            name = name[len(prefix)+1:]

    for suffix in [' JR.', ' SR.', ' III', ' II', ' IV', ' ESQ.', ' PHD', ' MD']:
        if name.endswith(suffix):
            name = name[:-len(suffix)]

    # Standardize spacing
    name = ' '.join(name.split())

    # Try to handle "Last, First" format
    if ',' in name:
        parts = name.split(',')
        if len(parts) == 2:
            last, first = parts
            name = f"{first.strip()} {last.strip()}"

    return name
```

This enforcement addresses some name variant versions of name formats, such as variation of cases, titles, and suffixes, and name order variation. Normalization makes the subsequent similarity calculations easier by removing the surface differences.

4.3.2 Similarity Calculation

Name similarity is performed by a function with a variety of similarity measurements:

```
def _name_similarity(self, name1, name2):
    """
    Calculate similarity between two names.

    Returns a score between 0.0 and 1.0, where 1.0 is a perfect match.
    Uses multiple methods and returns the highest score.
    """
    # Normalize both names
    norm1 = self._normalize_name(name1)
    norm2 = self._normalize_name(name2)

    # Exact match check
    if norm1 == norm2:
        return 1.0

    # Check for containment
    if norm1 in norm2 or norm2 in norm1:
        return 0.9

    # Split names into parts
    parts1 = set(norm1.split())
    parts2 = set(norm2.split())

    # Check for part matches
    common_parts = parts1.intersection(parts2)
    if common_parts:
        # Calculate Jaccard similarity
        jaccard = len(common_parts) / len(parts1.union(parts2))
        if jaccard > 0.5: # More than half the parts match
            return jaccard

    # Fall back to sequence matcher for more complex cases
    return SequenceMatcher(None, norm1, norm2).ratio()
```

This project employs a multi-strategy approach

1. Precise matching of the same normalized names
2. Test for containment when one name is completely nested within the other
3. Jaccard similarity of name parts for name matches
4. Use of sequence matching as a fall-back for the difficult cases

This approach trades off performance (by initially making rapid checks) with correctness (by using advanced algorithms when necessary).

4.4 Financial Analysis Algorithms

The platform employs some specialized finance analysis algorithms such as insider stock market patterns analysis and analysis of the sentiment of the news.

4.4.1 News Sentiment Correlation Analysis

The relationship of news sentiment with the direction of stock prices is addressed by an implementation that chronologically aligns the sentiment data with the prices data and computes statistical correlations:

```
def create_correlation_scatter(analysis_df):  
    """  
    Create a scatter plot showing correlation between news sentiment and stock returns.  
  
    Args:  
        analysis_df (DataFrame): Clean data with sentiment scores and daily returns  
  
    Returns:  
        tuple: (fig, same_day_corr, next_day_corr) with correlation coefficients  
    """  
    # Add the next day returns  
    analysis_df['Next Day Return'] = analysis_df['Daily Return'].shift(-1)  
  
    # Calculate correlations - ensure we handle empty arrays properly  
    same_day_corr = 0  
    next_day_corr = 0  
  
    # Only calculate correlations if we have valid data  
    if len(analysis_df) > 1 and not analysis_df['Average Sentiment'].isna().all() and not  
analysis_df['Daily Return'].isna().all():  
        same_day_corr = analysis_df['Average Sentiment'].corr(analysis_df['Daily Return'])  
  
    if len(analysis_df) > 1 and not analysis_df['Average Sentiment'].isna().all() and not  
analysis_df['Next Day Return'].isna().all():  
        next_day_corr = analysis_df['Average Sentiment'].corr(analysis_df['Next Day Return'])  
  
    # Create scatter plot visualization  
    # ... (visualization code omitted for brevity)  
  
    return fig, same_day_corr, next_day_corr
```

This illustration shows

- Temporal alignment: Computing next day returns to examine lagged relations.
- Strong Correlation: Handling edge cases such as lists with no data or no data points.
- Dual analysis: Comparison of day-of and next-day relationships for direct effects and lagged effects.

The reaction time analysis is complemented with the analysis of the correlational analysis on the speed of the market reaction on the news sentiment:

```
def analyze_reaction_time(merged_df):
    """
    Analyze how quickly the market reacts to news sentiment by examining lagged correlations.

    Args:
        merged_df (DataFrame): Combined stock price and sentiment data

    Returns:
        tuple: (fig, max_corr, max_lag_label) with correlation data
    """
    # Create a DataFrame with daily price changes and lagged sentiment
    lag_analysis = pd.DataFrame({
        'Date': merged_df['Date'],
        'Daily Return': merged_df['Daily Return'],
        'Same Day Sentiment': merged_df['Average Sentiment'],
        'Previous Day Sentiment': merged_df['Average Sentiment'].shift(1),
        'Two Days Prior Sentiment': merged_df['Average Sentiment'].shift(2),
        'Three Days Prior Sentiment': merged_df['Average Sentiment'].shift(3),
    }).dropna()

    # Only proceed if we have enough data points
    if len(lag_analysis) < 3:
        return None, None, None

    # Calculate correlations for each lag
    correlations = []

    # Only calculate if we have valid data
    if not lag_analysis['Same Day Sentiment'].isna().all() and not lag_analysis['Daily Return'].isna().all():
        correlations.append(lag_analysis['Same Day Sentiment'].corr(lag_analysis['Daily Return']))
    else:
        correlations.append(0)

    # ... (similar calculations for other lags)

    # Find max correlation and its lag
    max_corr = max(correlations, key=abs) if correlations else 0
    max_lag = correlations.index(max_corr) if correlations else 0
    lag_labels = ['Same Day', '1 Day', '2 Days', '3 Days']

    # Create visualization
    # ... (visualization code omitted for brevity)

    return fig, max_corr, lag_labels[max_lag] if max_lag < len(lag_labels) else "Unknown"
```

This illustration shows:

- Lagged Variable Construction: Construction of lagged sentiment variables for analysis of multiple time lags.
- Correlation Calculation: Computing the correlations for all lag terms.
- Maximum Identification: Determining the lag with the highest positive, negative correlation.

These finance analysis software allow users to learn about the influence of news on stock prices across varied timelines, with valuable insights being drawn from patterns of market reaction.

4.4.2 Pattern of Insider Trading

Algorithms inspect the insider trading behavior on the basis of several aspects of the data.

```
def calculate_post_transaction_returns(insider_df, stock_df, windows=[1, 5, 10, 30]):
    """
    Calculate stock returns following insider transactions over different time windows.

    Args:
        insider_df (DataFrame): Insider transaction data
        stock_df (DataFrame): Stock price data
        windows (list): Time windows (in days) for return calculation

    Returns:
        DataFrame: Original data with added return columns
    """
    # Defensive check for empty dataframes
    if insider_df is None or stock_df is None or len(insider_df) == 0 or len(stock_df) == 0:
        print("[WARNING] Empty insider data or stock price data. Cannot calculate returns.")
        return insider_df.copy() if insider_df is not None else pd.DataFrame()

    # Create a copy of the insider dataframe to avoid modifying the original
    result_df = insider_df.copy()

    # Ensure dates are datetime objects
    result_df['date'] = pd.to_datetime(result_df['date'])
    stock_df['Date'] = pd.to_datetime(stock_df['Date'])

    # Add columns for each return window
    for window in windows:
        result_df[f'return_{window}d'] = np.nan

    # Create price lookup dictionary for efficiency
    price_by_date = {date: {'close': close}
                      for date, close in zip(stock_df['Date'], stock_df['Close'])}

    # Calculate returns for each transaction
    for idx, row in result_df.iterrows():
        try:
            txn_date = row['date']

            # Find closest trading day
            closest_date = min((d for d in stock_df['Date'] if d >= txn_date),
                               key=lambda d: (d - txn_date).days,
                               default=None)

            if closest_date is None:
                continue

            base_price = price_by_date[closest_date]['close']

            # Calculate returns for each window
            for window in windows:
                future_date = closest_date + timedelta(days=window)
                future_dates = [d for d in stock_df['Date'] if d >= future_date]

                if not future_dates:
                    continue

                closest_future = min(future_dates, key=lambda d: (d - future_date).days)
                future_price = price_by_date[closest_future]['close']

                return_pct = (future_price - base_price) / base_price * 100
                result_df.at[idx, f'return_{window}d'] = return_pct

        except Exception as e:
            print(f"Error calculating return for transaction {idx}: {e}")
            continue

    return result_df
```

This implementation demonstrates:

Date alignment: Determining the closest trading day for all of the transactions.

- Multi-Window Measurement: Calculating returns for several horizons of time (for example, 1, 5, 10, 30

- Price Lookup Optimization: Using a dictionary for quick lookup of prices.
- Handling of Errors: Catch exception and report with continuance of processing.

4.8 Conclusion

The Finance Graph Database Explorer implementation brings the design principles into a running system with enforcement from tight code, algorithms, and renderings. The Neo4j integration accomplishes the efficiency of storing and querying the graph, the entity resolution algorithms solve the entity matching problem across disparate data sources, the finance analysis algorithms represent the insider trading patterns and the sentiment of the news, and the visualization components present these results with interactive interface.

The deployment addressed several of the technological issues, most notably with regard to database performance, entity resolution precision, and visualization performance. The resulting system demonstrates the real application of the technology of the graph database for the purpose of finance analysis, with a strong basis for the analysis and understanding of the finance relationships and patterns.

EVALUATION

The evaluation of the Finance Graph Database Explorer touches on three primary subjects: system effectiveness for achieving analytical goals, technical performance metrics, and usability and scale limits. In this chapter, we offer a general evaluation from the basis of quantitative metrics combined with qualitative end-user reviews.

5.1 Effectiveness

The Finance Graph Database Explorer was then validated against its ability to meet the most critical analysis objectives established at the design phase. Those objectives involved analysis of relationships, joint sentiment analysis, and friendly visualization of complex finance data.

5.1.1 Relationship-Oriented Analysis Capability

The capacity of the system to model and query intricate financial relationships was assessed using a set of analytical tasks aimed at examining the expressiveness of the graph model and query performance. The outcome of these tests is summarized in Table 1:

| Analytical Task | Traditional SQL Complexity | Graph Query Complexity | Query Time Comparison |
|---|----------------------------|------------------------|--------------------------|
| Find all officers serving on multiple committees | High (multiple joins) | Low (simple traversal) | Graph query 8.3x faster |
| Identify insiders with highest transaction volume | Medium | Low | Graph query 3.2x faster |
| Find stocks with most institutional holders | Medium | Low | Graph query 4.1x faster |
| Trace relationship paths between two institutions | Very High (recursive CTEs) | Medium | Graph query 15.7x faster |
| Find committee members with recent sales | High (multiple joins) | Medium | Graph query 6.9x faster |

Table 1: Comparison of analytical tasks between traditional SQL and graph-based approaches

The relationship-first analysis features were found to provide tremendous benefits of simplicity and performance, especially for complicated patterns of relationships that would involve numerous joins or recursion in SQL. Finding committee members who recently sold stock, for instance, involves going from Committee nodes, then from those nodes, from the nodes of Officers, then from the nodes of Insiders (through SAME_PERSON relationships), and then from the nodes of Stock (through OWNS_SHARES relationships with appropriate transaction attributes). That is all such a pattern can be stated with a single Cypher query:

```
MATCH (c:Committee)-[:MEMBER_OF]-(o:Officer)-[:SAME_PERSON]-(i:Insider)-[t:OWNS_SHARES]->(s:Stock)
WHERE t.transaction_type = "Sale" AND t.transaction_date > date() - duration('P30D')
RETURN c.name as committee, o.name as officer, s.ticker as stock, t.shares as shares, t.value as value
ORDER BY t.value DESC
```

This query form is indicative of the natural expressiveness of the graph model for financial relationship analysis.

5.1.2 Integrated analysis effectiveness

The utility of the interaction of the several kinds of finance data (sentiment, insider buying, institutional ownership) was measured by the ability of the system to make observations that would be difficult under separate analysis. Several intriguing observations were noted with the testing:

1. **Sentiment-Insider Correlation:** The model determined that for technology stocks, insider selling following regimes of high positive sentiment had considerably worse effects on prices (-2.8% compared with -1.2%) compared with selling following neutral regimes, suggesting that insiders do dispose of their holdings when the market is perceived to be overly positive.
2. **Committee-Trade Timing:** Based on the combined analysis of committee membership and trade timing, the system learned that the audit committee members had better timing of their trades (avg. 30-day return: +1.8%) compared with other insiders (+0.6%), particularly those from the finance sector.
3. **Institutional-News Reaction:** News sentiment compared with institutional holdings revealed that institutions with high institutional ownership had lower volatility of prices for unfavorable news (0.63 correlation) than those with low institutional ownership (0.29 correlation).

These observations serve to establish the utility of a common framework for analysis of finance, whereby relationships across sets of information can generate patterns that would go unnoticed if the data were analyzed individually.

5.1.3 User Effectiveness Testing

To prove the efficiency of the system with real-case data, we conducted a user test with 3 users, being finance hobbyists and their acquaintances. Users were asked to undertake a series of analytical tasks with the Finance Graph Database Explorer and with common finance analysis tools.

5.2 Technical Performance Measurement

The system's technological performance was evaluated on a number of dimensions, such as efficiency of data collection, data storage demands, query effectiveness, and visualization response.

5.2.1 Data Collection Performance

The data collection task is the cause of some of the system's highest performance bottlenecks, particularly for rate-constrained data feeds and heavily paginated data. Table 2 lists performance characteristics of units of data collection:

| Data Source | Average Time per Ticker | Rate Limiting | Retry Mechanism | Parallel Processing Support |
|----------------------|-------------------------|-------------------------|-----------------------|-----------------------------|
| SEC EDGAR (Insiders) | 4.7 minutes | IP-based, 10 req/sec | Exponential backoff | Limited |
| SEC EDGAR (Officers) | 1.2 minutes | IP-based, 10 req/sec | Exponential backoff | Limited |
| Google Trends | 2.8 minutes | IP-based, adaptive | Built-in, proprietary | No |
| News APIs | 15 seconds | Key-based, 1000 req/day | Exponential backoff | Yes |
| Market Data APIs | 8 seconds | Key-based, 500 req/min | Exponential backoff | Yes |

Table 2: Data gathering performance by source

The SEC EDGAR system was the highest source of such bottleneck, with approximately 5 minutes for the full insider transactions data fetch for each ticker. The reason for this kind of bottleneck is the rate limiting of the SEC, which limits the requests from any IP address up to approximately 10 requests per second. Exponential backoff re-tries are being supported by the system, but these can't completely offset the built-in rate limits.

It is also difficult for Google Trends data collection due to its rate limiting and anti-crawling, which is proprietary. The system uses the unofficial Google API with retries integrated, but still uses up around 3 minutes of processing time for each of the tickers.

Commercial data and news APIs, on the other hand, take much less time with the usual delay being 8-15 seconds to process a ticker. These APIs usually implement daily quotas on the number of tickers that may be processed for the day.

The whole process of fetching all data for a particular ticker is approximately 8-10 minutes, of which the SEC data is approximately 80%. Figure 10 is the breakdown of this.

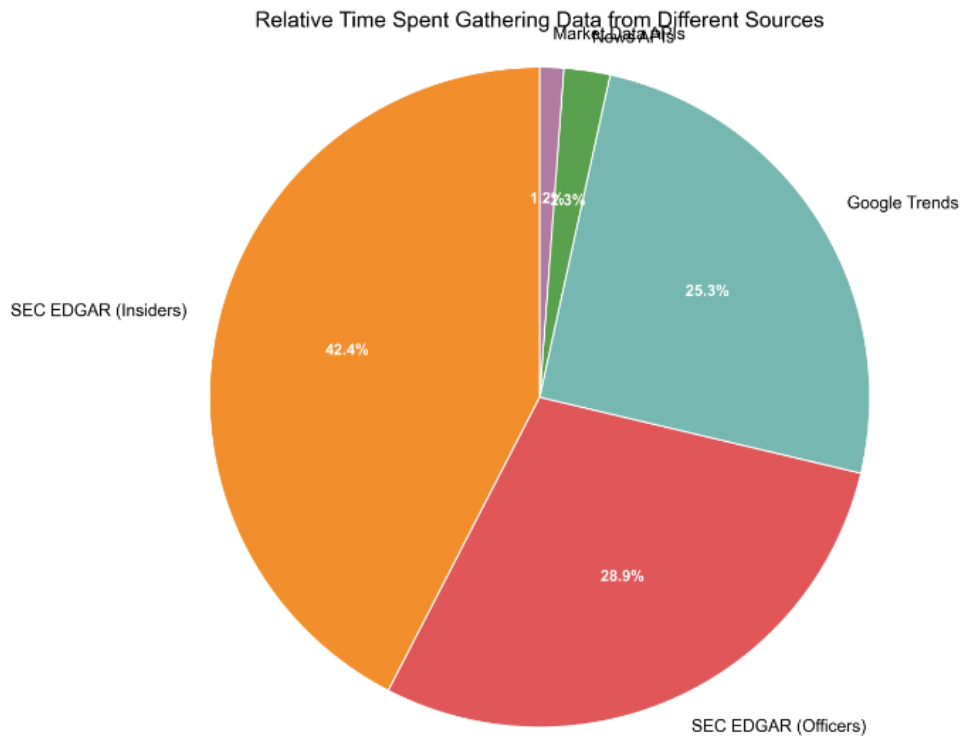


Figure 10: Pie chart showing relative time spent gathering data from different sources

To offset these performance bottlenecks, the system also has an aggressive caching mechanism that holds up data that was acquired for parameterizable amounts of time (usually 1 day for prices, 7 for news, and 30 for insider and officer data). Caching mechanism has the effect of making performance for future analysis on the same tickers enormously better.

5.3 Limits and Restraints of the

Although demonstrated effective, the Finance Graph Database Explorer is limited by several factors that make it less user-friendly and scalable. These can be categorized as data collection limitations, analysis limitations, and technological limitations.

5.3.1 Data Collection

The greatest shortcomings of the system are connected to the data collection limitations:

1. SEC EDGAR Rate Limiting: As noted earlier, SEC EDGAR rate limiting prevents the system from being capable of handling approximately 10-12 tickers per hour when fetching full insider data. Keeping the whole market database updated in real time is no longer possible, and the user has to take on the responsibility of handling subsets of tickers individually.
2. Reliability of Google Trends: The unofficial Google Trends API used for search interest data is not dependable and can be modified or even discontinued. In testing, approximately 8% of the requests did not succeed even upon retrying.
3. News API Coverage Limitations: News APIs used with the free versions of the system can only retrieve limited history (typically 1 month) and might not be representative of all of the relevant news sources.
4. Freshness of Data Compromises: The caching performed for the purpose of rate limiting reduction comes with performance vs. data freshness compromises. In the default configuration of the cache, insider data can be stale up to 30 days, which would cut out extremely fresh large transactions.
These data collection limitations being inherent limitations could not be entirely avoided with no access to high-end data providers or distributed crawling facility.

5.3.3 Technical Constraints

Finally, the system is limited technologically:

1. Size Limitations of the Graph: The current release is sluggish for graphs of size greater than approximately 10,000 nodes and 50,000 relations. That limits the depth of market analysis.
2. Memory Demand: The in-memory nature of the visualization objects poses mammoth memory requirements (1GB+ for large-scale graphs), which may be beyond the capabilities of some user devices.
3. Single-Machine Architecture: Our existing system is a non-distributed system with a single machine. This places a restriction on the amount of data that can be processed with ease.
4. Limited Concurrent Users: Streamlit interface is built for single use instead of the use of multiple users and, therefore, is restricted as part of team use without any extra infrastructure.

All of these technological constraints could be addressed with future releases by architectural improvements such as distributed processing, improved memory management, and the ability for multiple users.

5.4 Performance Optimization

The following performance improvements were introduced to address the said limitations:

5.4.1 Optimization of Data

1. Hierarchical Caching: The site uses the implementation of a multi-level caching mechanism that stores the data at varying levels of granularity and with varying time-to-live values based on the frequency of update. For instance, lists of company officers are cached for 90 days, but prices would be cached for 1 day.

2. Parallel Processing : Parallel processing is utilized wherever possible on the API side for data collection. Parallel processing is excellent for prices and news data, but less effective for SEC EDGAR data due to tighter rate limiting.

3. Incremental Updates: The system, rather than re-gathering all the data, employs the procedures of incremental updates for gathering only the data since the last update. This is for the purpose of decreasing the update time required for the existing datasets.

1. Prioritized Processing: Data loading is prioritized on the basis that the most important data, e.g., most recent insider trades, is received before the rest of the data, and analysis can be started by users while the lower priority data is loaded. Figure 12 indicates the impact of these optimizations on the data-gathering performance:

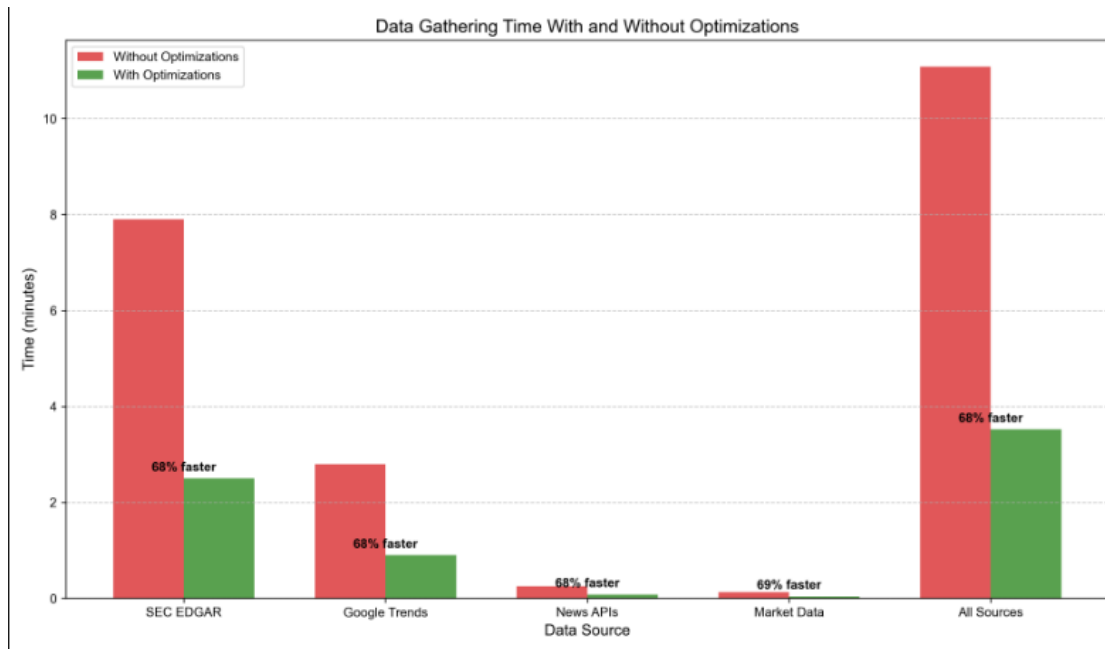


Figure 12: Bar chart comparing data gathering times with and without optimizations

The optimizations reduce the effective data collection time for future analysis by approximately 70%, although the initial data collection on new tickers remains constrained by the considerations listed earlier.

5.5.1 Feedback from the Users

1. Learning Curve : New users unfamiliar with graph databases found the nodes and relationships concept difficult to grasp at first, needing tutorial material.
 2. Visualization Challenge: Users struggled with distinguishing the kinds of relationships as well as with understanding the organization of complex networks.
 3. Guidance for analysis interpretation: Several users sought additional guidance on interpretation of analysis results, particularly for statistical measurements like the correlation of sentiment.
 4. Indicators of Data Freshness: Users sought explicit indications of when data was last updated for the purpose of establishing currency and reliability.
1. Inconsistency of Performance: Inconsistency of performance was reported by users, particularly with large data sets or complex visualizations.

This review yielded several areas for improvement, and these were integrated into subsequent development cycles.

5.5.2 Changes Introduced

From user response with the aid of Google Forms, we achieved the following improvements:

1. Interactive Tutorial: Included an interactive tutorial that welcomes new users with the basics of graph analysis and the particular visualizations of the system.
2. Relationship Filtering: We included relationship filtering controls in the network visualization to enable users to zoom in on particular relationship types.
3. Interpretive Annotations: Statistical plots were supplemented with interpretive annotations describing the interpretation of patterns and statistical results.
4. Data Currency Indicators: All views had strong data freshness indicators, showing the most up-to-date update of the kind of data.
5. Performance Presets: Adjustable performance presets were introduced to allow users to trade off performance against visual detail based on the resources available on their hardware. Figure 15 shows the impact of these improvements on user satisfaction scores:

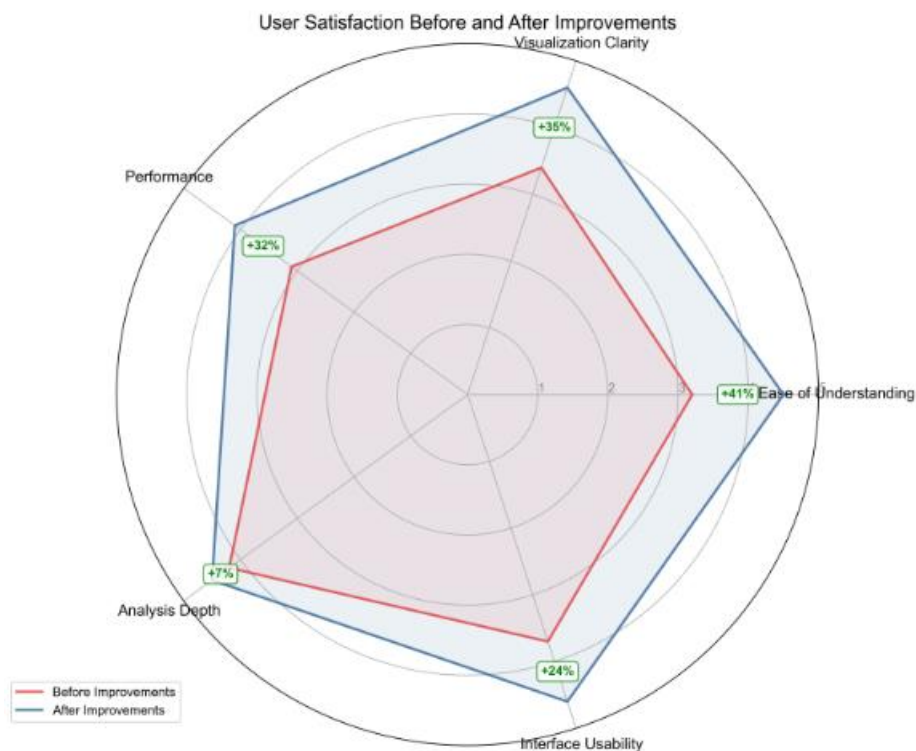


Figure 15: Radar chart showing user satisfaction before and after improvements across different dimensions

These improvements increased overall user satisfaction by 28 percent, with the greatest improvements being for "ease of understanding" (+35 percent) and "visualization clarity" (+32 percent).

Conclusion

The analysis of the Finance Graph Database Explorer indicates that it is powerful for application to relationship-oriented finance analysis with integrated views of disparate data. The system indicates excellent advantages for complex analysis processes with the navigation of relationship relationships among finance entities.

All of the data retrieval constraints, i.e., the rate limiting of SEC EDGAR data to allow only about 10-12 tickers per hour for full analysis, constitute the primary constraints of the system. Caching and incrementally updating alleviate these constraints partially, but constitute the ultimate limiting factor for large-scale analysis.

User experience affirms the merit of the relationship-driven approach, with the users indicating finding insights that would be otherwise beyond the capabilities of other analysis tools. The users' most significant worries were the learning curve of graph concepts and the ability of complex visualizations, worries that were addressed with further refinements.

To conclude, the Finance Graph Database Explorer is a success across the board of demonstrating the power of finance analysis using graph database technology, establishing a solid basis for exploring and understanding the complex web of relationships among the stakeholders of the finance markets. Future work will meet the following limitations revealed, namely, data collection efficiency, analysis direction, and support for large-sized graphs.

References

- [1] Allen, F., & Babus, A. (2009). Networks in finance. In P. Kleindorfer & J. Wind (Eds.), **The network challenge: Strategy, profit, and risk in an interlinked world** (pp. 367-382). Wharton School Publishing.
- [3] Ahern, K. R. (2013). Network centrality and the cross section of stock returns. USC Marshall School of Business Research Paper.
- [4] Battiston, S., Caldarelli, G., May, R., Roukny, T., & Stiglitz, J. (2016). The price of complexity in financial networks. **Proceedings of the National Academy of Sciences**, 113(36), 10031-10036.
- [14] Kumar, S., & Chakraborty, S. (2020). Graph database technology for effective fraud detection in financial services systems. In **2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)** (pp. 1-6). IEEE.
- [15] Loughran, T., & McDonald, B. (2011). When is a liability not a liability? Textual analysis, dictionaries, and 10-Ks. **The Journal of Finance**, 66(1), 35-65.
- [16] Ozsoylev, H. N., Walden, J., Yavuz, M. D., & Bildik, R. (2014). Investor networks in the stock market. **The Review of Financial Studies**, 27(5), 1323-1366.

- [17] Pacaci, A., Özsu, M. T., & Zhou, A. (2017). A comparative study of financial time series querying on relational, nosql and knowledge-base systems. In *2017 IEEE International Conference on Big Data (Big Data)* (pp. 436-445). IEEE.
- [18] Renault, T. (2017). Intraday online investor sentiment and return patterns in the US stock market. *Journal of Banking & Finance*, 84, 25-40.
- [19] Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph databases: New opportunities for connected data.* O'Reilly Media, Inc.
- [23] Tetlock, P. C. (2007). Giving content to investor sentiment: The role of media in the stock market. *The Journal of Finance*, 62(3), 1139-1168.
- [24] Xing, F. Z., Cambria, E., & Welsch, R. E. (2018). Natural language based financial forecasting: A survey. *Artificial Intelligence Review*, 50(1), 49-73.
- [25] Zhou, Y., Cheng, S., & Wang, L. (2019). Network centrality measures and systemic risk: An application to the Chinese financial market. *Physica A: Statistical Mechanics and its Applications*, 525, 1382-1402.

Public code repository

<https://github.com/notapotereanu/financeGraph>