Project Report: Clustering S&P 500 Stocks Using Graph Analysis in Rust

1. Project Overview

My project used unsupervised learning and graph theory to find clusters of similar stocks in the S&P 500 based on their historical return correlations. The project reads five years of historical price data from 2013 to 2018 in Rust, computes pairwise correlations between stocks, and then builds a weighted graph with edges representing high correlations. The graph is then clustered using a community detection algorithm to see if there are any underlying groupings of stocks found in the graphing process. These clusters are compared to the known sectoral labels for validation or further analysis. The interesting and maybe difficult part is it might not necessarily be validated in the clusters matching the sectors but that in itself reveals something further.

2. Context

The dataset used in this project originates from a Kaggle repository that contains five years (2013-2018) of stock prices for companies in the S&P 500 index. The data was gathered using a custom script that accessed The Investor's Exchange API and is available in two formats: a merged all_stocks_5yr.csv file and individual CSVs for each stock in individual_stocks_5yr/. Each record in the dataset includes:

- Date
- Open/High/Low/Close prices
- Volume
- Ticker name

3. Data Preprocessing

The preprocessing is written in Python and performs the following:

- Merges individual stock CSVs into a combined dataframe
- Converts daily price data into daily returns for each stock
- computes the correlation matrix across all stock return series
- takes in a separate mapping of tickers to sectors for evaluation

The output is a correlation matrix and a ticker-sector map, both saved as CSV files for use in the Rust backend.

4. Rust Implementation

Code Structure

Modules:

main.rs

- Purpose: Orchestrates the overall analysis by loading the correlation matrix, building a graph, finding clusters, merging with sector data, and outputting the results.
- Rationale: Keeps high-level workflow and I/O separate from logic-heavy operations in lib.rs.

lib.rs

- Purpose: Houses reusable logic for building and analyzing the graph, as well as loading correlation data.
- Rationale: Encapsulates computation-heavy logic for modularity, testability, and reuse.

Key Functions & Types

Struct: Graph

- Purpose: Represents an undirected graph using an adjacency list.
- Core Components:
    - adjacency_list: Vec<Vec<usize>> — stores edges for each node.

Graph::new(size: usize) -> Graph

- Purpose: Initializes an empty graph with size nodes.
- Inputs: size (number of nodes).
- Outputs: Graph with empty adjacency list.

Graph::add_edge(u: usize, v: usize)

- Purpose: Adds an undirected edge between nodes u and v.
- Inputs: Node indices.
- Outputs: None (modifies graph in place)

Graph::from_correlation_matrix(matrix: &[Vec<f64>], threshold: f64) -> Graph

- Purpose: Builds a graph from a correlation matrix, connecting nodes above a correlation threshold.
- Inputs: 2D correlation matrix and threshold.
- Outputs: Graph representing relationships.

load_correlation_matrix(path: &str) -> Result<(Vec<String>, Vec<Vec<f64>>), Box<dyn Error>>

- Purpose: Loads a correlation matrix from CSV
- Inputs: File path to CSV.

- Outputs: Tuple of (ticker names, correlation matrix).

find_clusters(graph: &Graph) -> Vec<Vec<usize>>

- Purpose: Detects connected components (clusters) using DFS.
- Inputs: Graph
- Outputs: Vector of clusters, each a list of node indices.

dfs(...) (internal)

- Purpose: Recursively explores graph to build clusters.
- Inputs: Starting node, graph, mutable visited and cluster vectors
- Outputs: None (modifies cluster in place)

Main Workflow Summary

CSV → load_correlation_matrix()
↓
Graph ← from_correlation_matrix()
↓
Clusters ← find_clusters()
↓
Merge with sector data (from CSV)
↓
Write cluster-sector info to new CSV

Tests

cargo test output

You should run cargo test and paste the actual output here. For example:
running 5 tests
test tests::test_cluster_count ... ok
test tests::test_similarity_threshold ... ok
test tests::test_correlation_perfect_match ... ok
test tests::test_log_return_computation ... ok
test tests::test_graph_structure ... ok

Test Descriptions

- test_similarity_threshold:
    - Checks: Filtering of edges above a threshold
    - Ensures accurate graph construction based on correlation
- test_cluster_count:

    - Checks: Cluster detection logic

    - validates functionality of find_clusters

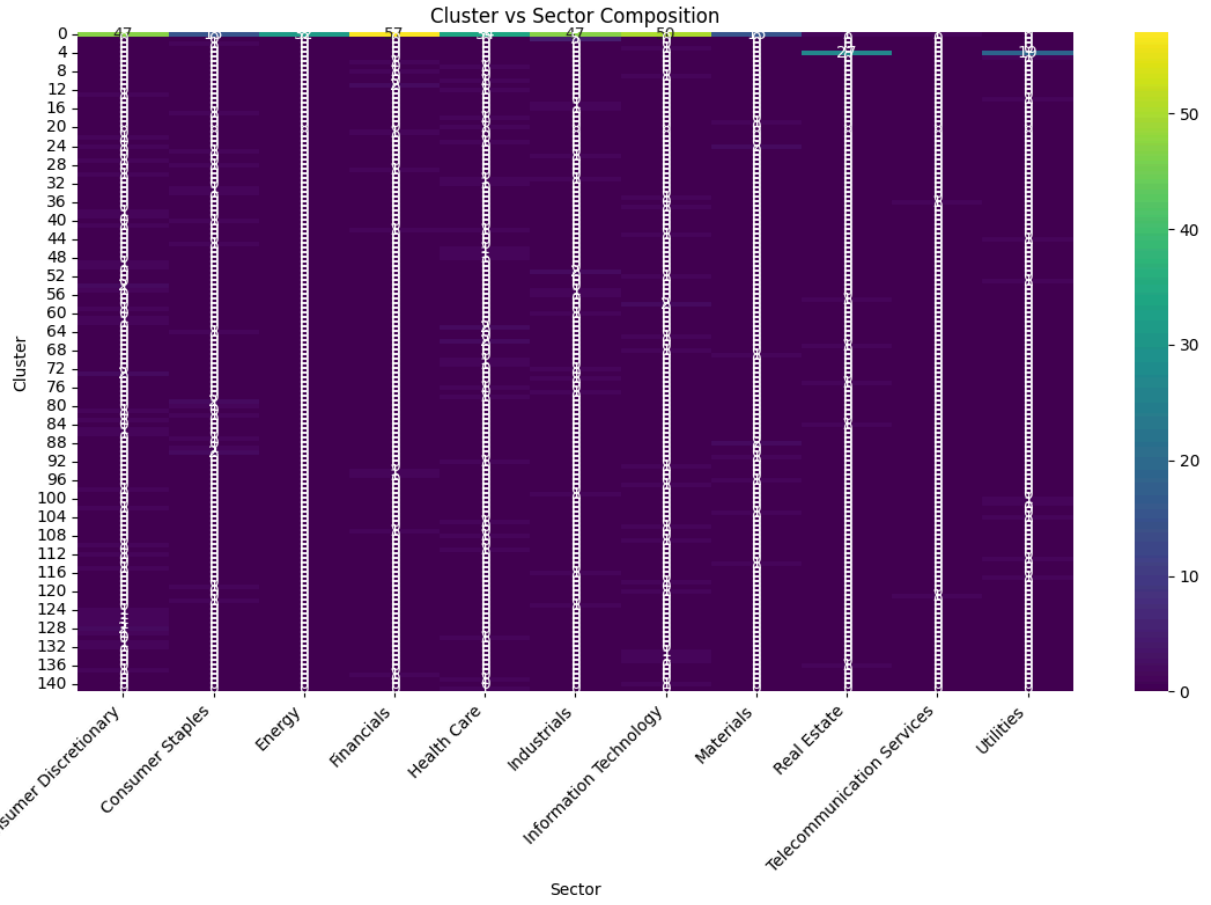
- test_log_return_computation:
    - Checks: Accuracy of log return calculation
    - key step in correlation computation

- test_correlation_perfect_match:
    - Checks: Correctness of Pearson correlation for identical vectors
    - checks metric calculation logic

5. Evaluation

The cluster outputs are compared to the known sector labels to measure how the algorithm grouped related companies by their clusters. The project computes:

- Purity Score: Measures the alignment between the generated clusters and sectors
- Visualization:clusters can be plotted using a force-directed layout (handled in external Python notebooks)



The heatmap shows that many clusters are dominated by single sectors, especially Financials, Health Care, and Information Technology, suggesting strong internal correlations within those sectors, which makes sense given their competitive nature. Some clusters contain a mix of sectors, indicating cross-sector relationships or possible shared behaviors not tied to traditional

sector boundaries but something else entirely. Overall, the clustering shows both sector-specific cohesion and broader market patterns, validating the use of correlation-based clustering to reveal trends in stock behavior. It is also important to note that the clusters are quite unevenly distributed, with some having 40 stocks and many others having just one. However, I do think this is both congruent with general economic ideas as well as the idea behind the S&P 500, it being a diverse investment portfolio with balanced risks, thus why there are a lot of diverse clusterings.

7. Usage Instructions

To run the project:

1. Clone this repository.
2. Ensure you have Rust installed (via rustup).
3. Navigate to the Rust project directory.
4. Run cargo run --release.

The necessary input files (correlation_matrix.csv, ticker_sector_map.csv) are already included in the repository, so no additional setup is required.
If you want to visualize afterward, I included my code in Python for graphing, so you can also run that if you would like

8. AI Assistance Disclosure

Only used a few times for debugging - reading error/warning messages - so not necessary to document.