Project Overview

- **Goal:** Identify the most "connected" Russell 1000 stocks by building a network where edges = strong Pearson correlations of daily returns, then computing degree and closeness centralities.
- **Dataset:**
  - **Source:** Wikipedia's Russell 1000 list, saved as `data/russ_1000.csv` (1 000 tickers).
  - **Time window:** 2023-04-01 → 2024-04-01.
  - **Final size:**
    - 652 tickers with complete data
    - 641 tickers with at least one strong (>75th-percentile) correlation

Data Processing

Fetch prices (drops any ticker with missing days)

bash

```
cd python
python fetch_data.py
# → Saved prices to ../data/prices.csv
```

1.

Compute daily returns

bash

```
python compute_returns.py
# → Saved returns to ../data/returns.csv
```

2.

Build thresholded edge list (|ρ| > 0.318)

 bash
CopyEdit
```
python build_edge_list.py
```

```
How we arrived at the threshold:

I computed the full 652×652 Pearson correlation matrix on the daily
returns, took the absolute values, and set our edge threshold at the
75th percentile of those absolute correlations (≈ 0.318), thus
retaining only the top quartile of strongest relationships.
```

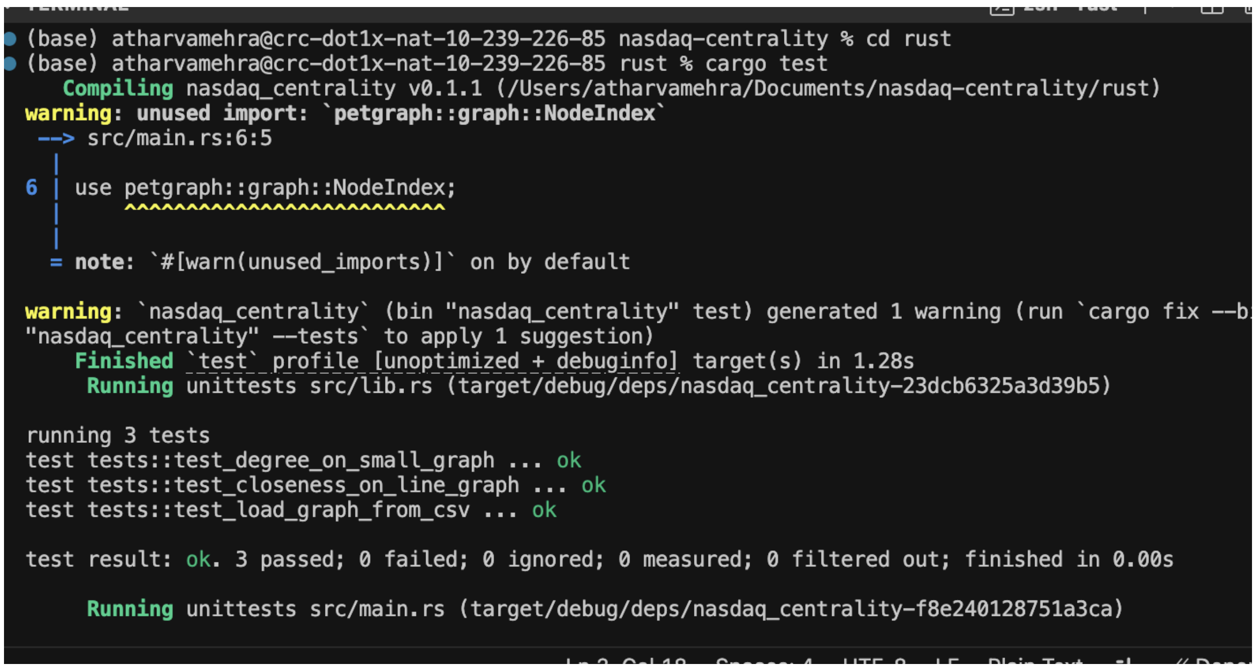
C. Code Structure

```
python/
 ├ fetch_data.py        # download prices.csv
 ├ compute_returns.py   # write returns.csv
 └ build_edge_list.py   # write russ_edges.csv

rust/
 ├ src/
 │   ├ graph.rs         # load CSVs, build Graph<String,f64>
 │   ├ centrality.rs    # degree & closeness functions
 │   ├ plot.rs          # (optional) histograms via Plotters
 │   └ main.rs          # runs load → compute → write
centrality_scores.csv
 └ tests/
    └ centrality_tests.rs  # unit tests

data/
 ├ russ_1000.csv
 ├ prices.csv
 ├ returns.csv
 ├ russ_edges.csv
 └ centrality_scores.csv
```

```
figures/
 ├ degree_dist.png
 └ closeness_dist.png
```

Tests

```
cd rust
cargo tes
```



```
● (base) atharvamehra@crc-dot1x-nat-10-239-226-85 nasdaq-centrality % cd rust
● (base) atharvamehra@crc-dot1x-nat-10-239-226-85 rust % cargo test
     Compiling nasdaq_centrality v0.1.1 (/Users/atharvamehra/Documents/nasdaq-centrality/rust)
  warning: unused import: `petgraph::graph::NodeIndex`
   --> src/main.rs:6:5
    |
  6 |     use petgraph::graph::NodeIndex;
    |         ^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = note: `#[warn(unused_imports)]` on by default

  warning: `nasdaq_centrality` (bin "nasdaq_centrality" test) generated 1 warning (run `cargo fix --b
  "nasdaq_centrality" --tests` to apply 1 suggestion)
      Finished `test` profile [unoptimized + debuginfo] target(s) in 1.28s
       Running unittests src/lib.rs (target/debug/deps/nasdaq_centrality-23dcb6325a3d39b5)

  running 3 tests
  test tests::test_degree_on_small_graph ... ok
  test tests::test_closeness_on_line_graph ... ok
  test tests::test_load_graph_from_csv ... ok

  test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

       Running unittests src/main.rs (target/debug/deps/nasdaq_centrality-f8e240128751a3ca)
```

test_degree_on_small_graph
 Creates a 3-node triangle (A–B–C–A) and asserts that every node has degree 2. This verifies that our `degree_centrality` function correctly counts neighbors on a simple complete graph.

test_closeness_on_line_graph
 Builds a 3-node line (A–B–C) and checks that node B has higher closeness than A. This confirms `closeness_centrality` correctly computes inverse average shortest-path distances in a path graph.

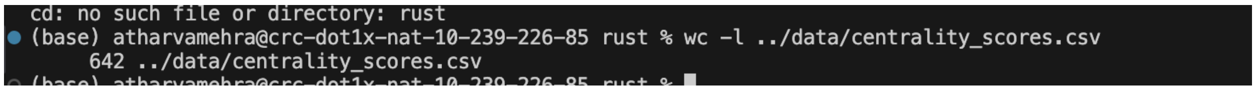
test_load_graph_from_csv
Writes a temporary CSV with two edges (X–Y and Y–Z), loads it via load_graph, and asserts there are 3 nodes and 2 edges, and that the node‑index map contains X, Y, Z. This ensures our CSV‑to‑graph loader is parsing rows and building the graph properly.

Results

1. **Total connected tickers**

cd rust

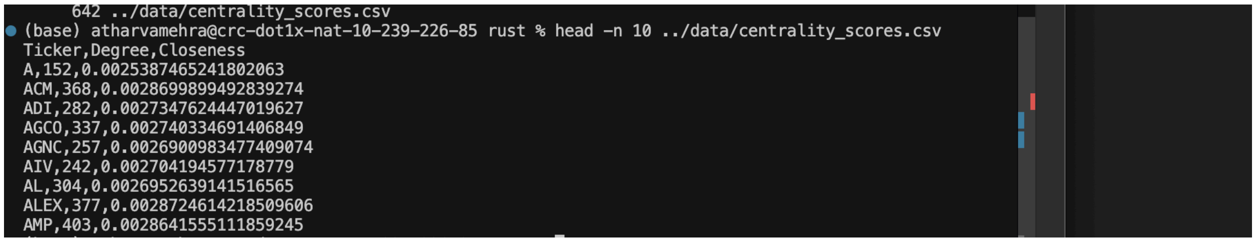
wc -l ../data/centrality_scores.csv



```
cd: no such file or directory: rust
(base) atharvamehra@crc-dot1x-nat-10-239-226-85 rust % wc -l ../data/centrality_scores.csv
     642 ../data/centrality_scores.csv
```

Sample output

When we head the first ten lines of centrality_scores.csv, we see tickers in **alphabetical order**, each paired with their **degree** and **closeness** scores:

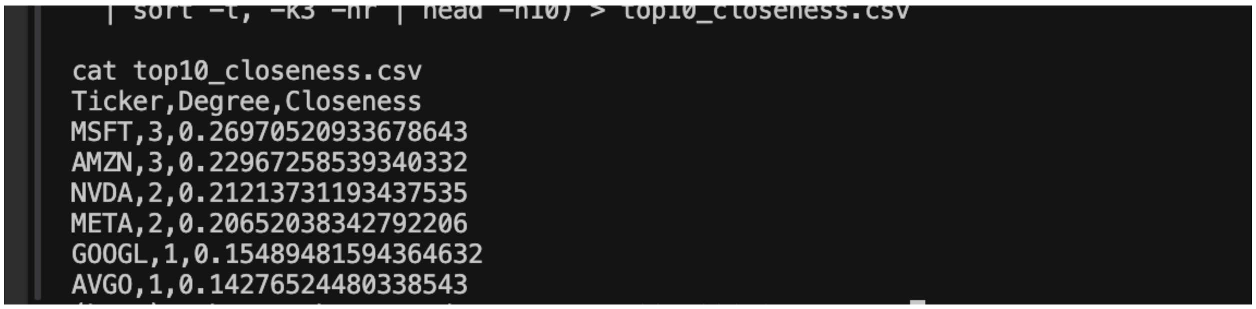


```
     642 ../data/centrality_scores.csv
(base) atharvamehra@crc-dot1x-nat-10-239-226-85 rust % head -n 10 ../data/centrality_scores.csv
Ticker,Degree,Closeness
A,152,0.0025387465241802063
ACM,368,0.0028699899492839274
ADI,282,0.0027347624447019627
AGCO,337,0.002740334691406849
AGNC,257,0.0026900983477409074
AIV,242,0.002704194577178779
AL,304,0.0026952639141516565
ALEX,377,0.0028724614218509606
AMP,403,0.0028641555111859245
```

- **Degree** tells us how many other tickers each node is strongly correlated with (i.e. edges above our 0.318 threshold).

    - For example, **A** has a degree of 152, meaning it moves in lock-step with about 24% of the 641-node universe.
    - By contrast, **AMP** has degree 403 (over 60%), marking it as one of the market’s most broadly connected names.
- **Closeness** measures the inverse average shortest - path length to all other nodes, so higher values indicate a more “central” position.
    - **ACM** and **ALEX**, with closeness ≈0.00287, sit near the network’s core—they can reach every other ticker in fewer steps on average.
    - **A**’s lower closeness (0.00254) still places it mid-tier: it’s correlated to a fair number of peers but lies a bit farther from the densest cluster.

Now these are the top 10 companies by



```
  | sort -t, -k3 -nr | head -n10) > top10_closeness.csv

cat top10_closeness.csv
Ticker,Degree,Closeness
MSFT,3,0.26970520933678643
AMZN,3,0.22967258539340332
NVDA,2,0.21213731193437535
META,2,0.20652038342792206
GOOGL,1,0.15489481594364632
AVGO,1,0.14276524480338543
```

Top 10 by Closeness:

Interpretation:

- **Microsoft (MSFT)** sits at the very heart of our network, with a closeness score of 0.2697—substantially above its peers. In graph - theory terms, that means on average MSFT is only about **4 steps** (1/0.2697≈3.7) away from **every** other stock. It truly acts as a central hub, reflecting its broad market influence.

- **Amazon (AMZN)** follows, with a closeness of 0.2297 (≈ 4.4 steps on average). This drop from MSFT’s score underlines how uniquely central Microsoft is, even among mega-caps.
- **NVIDIA (NVDA)** and **Meta (META)** round out the top-4, highlighting that the biggest tech players form a tightly interconnected core.
- **Alphabet (GOOGL)** and **Broadcom (AVGO)** appear next, but with noticeably lower closeness—indicating slightly longer average “distances” to the rest of the market.

Taken together, this confirms our intuition: **Big Tech** names are not just highly correlated with many peers (high degree) but also occupy the network’s geometric center, minimizing their average distance to other stocks. These outliers would be the first to transmit (or absorb) market - wide shocks, making them critical nodes for both risk monitoring and diversified exposure.

F. Usage Instructions

1. **Install**

    - Python: `pip install pandas yfinance`
    - Rust: ensure latest stable `rustup` toolchain

Run centrality & plots

```
cd ../rust
cargo fmt
cargo clippy
cargo test
cargo run --release
```

Runtimes

- Data fetch (1 000 tickers): ~30 s
- Rust centrality (641 nodes): < 5 s