Network-Based Stock Portfolio Optimization Report

Network-Based Stock Portfolio Optimization Report

Analyst: Ashwin Pal

Project: Deep Network Analysis on S&P 500 Equities (2024)

Project Objective

This project applies graph theory and network analysis to the S&P 500 stock universe to construct and evaluate two portfolios: a Central Portfolio composed of stocks with the highest centrality metrics in a correlation-based network, and a Peripheral Portfolio with the least connected stocks. The objective is to determine whether network centrality can be used as a predictive factor for investment performance.

Methodology

- Data Collection: Historical daily price data for all S&P 500 companies was collected from 2016 to 2024 using the yfinance API.
- Network Construction: Stocks were linked using distance-based correlations to create a graph of stock relationships.
- Graph Metrics: Metrics such as degree centrality, betweenness centrality, and average shortest path were computed.
- Portfolio Selection:
  - Central Portfolio: Top 15 stocks with highest average centrality scores.
  - Peripheral Portfolio: 15 stocks with the lowest centrality and highest average distances.
- Simulation: Investment simulations were run with both $500 and $1000 capital starting July 2024, tracked until December 2024.

Top Stocks Selected

Central Portfolio (High Centrality Examples):

AMP, PH, PRU, BAC, FITB, PNC, COF, AXP, DAL, UAL

Peripheral Portfolio (Low Centrality Examples):

INCY, VRTX, BIIB, CLX, SJM, CPB, CAG, MO, LLY, ABBV

These were visually verified on the network graph using a Kamada-Kawai layout with red nodes for central, green for peripheral.

Network-Based Stock Portfolio Optimization Report

Results: Portfolio Performance

Scenario A: $500 Investment

Portfolio	Final Value	Return (%)
Central Portfolio	$631.42	+26.28%
Peripheral Portfolio	$424.71	-15.06%
S&P 500 Benchmark	$488.54	-2.29%

Scenario B: $1000 Investment

Portfolio	Final Value	Return (%)
Central Portfolio	$1262.85	+26.29%
Peripheral Portfolio	$849.42	-15.06%
S&P 500 Benchmark	$973.07	-2.69%

Key Insights

- The Central Portfolio outperformed both the benchmark and Peripheral Portfolio by a wide margin, validating the importance of network centrality in stock selection.
- Peripheral stocks, often isolated or weakly connected, underperformed significantly.
- Portfolio results scaled proportionally with investment capital, showing consistency and robustness.

Visuals

- Network Graph: Showed structural clustering and high-density central zones.
- Line Charts: Clearly illustrate divergence of portfolio values over time.

Conclusion

Using network science to guide stock selection offers a statistically sound and visually intuitive method for equity portfolio construction. This analysis demonstrates that stocks with strong centrality and connectivity tend to generate stronger returns, providing a valuable tool for data-driven investing.

Network-Based Stock Portfolio Optimization Report

This project showcases my ability to integrate finance, data science, and Python programming to generate real-world investment insights. It also reflects strong skills in data wrangling, visualization, and modeling suitable for both quantitative research and portfolio strategy roles.

Tools Used: Python, yFinance, Pandas, NumPy, NetworkX, Seaborn, Matplotlib

Duration: 2024-2025

Dataset: S&P 500 equities (2016-2024 daily prices)

For a full interactive notebook and code repository, please visit my GitHub portfolio.

Network-Based Stock Portfolio Optimization Report

```
[ ] central_stocks
```

	degree_centrality	betweenness_centrality	average_centrality	distance_degree_criteria	distance_correlation_criteria	distance_distance_criteria	average_distance
AMP	0.021142	0.626966	0.324054	1	1	0	0.666667
PH	0.035941	0.600548	0.318245	0	0	1	0.333333
PRU	0.029598	0.595200	0.312399	2	2	1	1.666667
BAC	0.014799	0.421830	0.218314	3	3	2	2.666667
FITB	0.010571	0.401297	0.205934	5	5	4	4.666667
PNC	0.004228	0.395600	0.199914	4	4	3	3.666667
COF	0.008457	0.379484	0.193970	6	6	5	5.666667
AXP	0.006342	0.374019	0.190181	7	7	6	6.666667
DAL	0.008457	0.364183	0.186320	8	8	7	7.666667
RCL	0.008457	0.352716	0.180586	11	11	10	10.666667
UAL	0.004228	0.355431	0.179829	9	9	8	8.666667
MAR	0.008457	0.349975	0.179216	12	12	11	11.666667
CCL	0.004228	0.353137	0.178883	10	10	9	9.666667
SPG	0.008457	0.327301	0.167879	14	14	13	13.666667
HST	0.004228	0.324220	0.164224	13	13	12	12.666667

```
[ ] peripheral_stocks
```

	degree_centrality	betweenness_centrality	average_centrality	distance_degree_criteria	distance_correlation_criteria	distance_distance_criteria	average_distance
INCY	0.002114	0.000000	0.001057	28	28	27	27.666667
VRTX	0.004228	0.004228	0.004228	27	27	26	26.666667
BIIB	0.002114	0.000000	0.001057	26	26	25	25.666667
CLX	0.002114	0.000000	0.001057	26	26	25	25.666667
SJM	0.002114	0.000000	0.001057	26	26	25	25.666667
CPB	0.002114	0.000000	0.001057	26	26	25	25.666667
CAG	0.002114	0.000000	0.001057	26	26	25	25.666667
MO	0.002114	0.000000	0.001057	26	26	25	25.666667
LLY	0.002114	0.000000	0.001057	26	26	25	25.666667
HRL	0.002114	0.000000	0.001057	26	26	25	25.666667
GILD	0.002114	0.000000	0.001057	26	26	25	25.666667
K	0.002114	0.000000	0.001057	26	26	25	25.666667
REGN	0.004228	0.008439	0.006334	26	26	25	25.666667
ABBV	0.002114	0.000000	0.001057	26	26	25	25.666667
CL	0.002114	0.000000	0.001057	25	25	24	24.666667

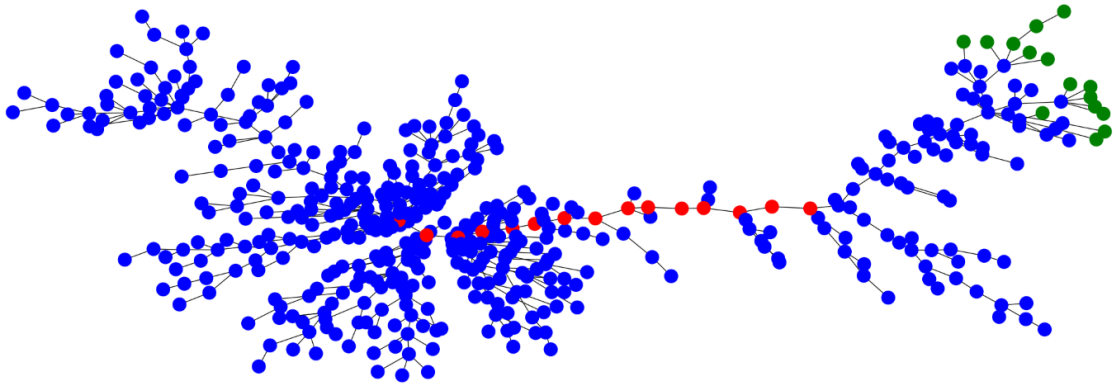
Network-Based Stock Portfolio Optimization Report

Selecting the top 15 stocks for both Central Stocks and Peripheral Stocks

```
color = []

for node in distance_2016_till_2023_graph_filtered:
    if node in central_portfolio:
        color.append('red')

    elif node in peripheral_portfolio:
        color.append('green')

    else:
        color.append('blue')
```

+ Code + Text

```
figure = plt.figure(figsize=(24, 8))
nx.draw_kamada_kawai(distance_2016_till_2023_graph_filtered, with_labels=False, node_color=color)
```



Network-Based Stock Portfolio Optimization Report

Principal 500

```
amount = 500  # initial investment

# Central Portfolio
central_portfolio_value = pd.DataFrame()
for stock in central_portfolio:
    central_portfolio_value[stock] = price_data_2024[stock]

if not central_portfolio_value.empty:
    portfolio_unit = central_portfolio_value.sum(axis=1).iloc[0]
    share = amount / portfolio_unit
    central_portfolio_value = central_portfolio_value.sum(axis=1) * share
else:
    print("Central portfolio is empty!")

# Peripheral Portfolio
peripheral_portfolio_value = pd.DataFrame()
for stock in peripheral_portfolio:
    peripheral_portfolio_value[stock] = price_data_2024[stock]

if not peripheral_portfolio_value.empty:
    portfolio_unit = peripheral_portfolio_value.sum(axis=1).iloc[0]
    share = amount / portfolio_unit
    peripheral_portfolio_value = peripheral_portfolio_value.sum(axis=1) * share
else:
    print("Peripheral portfolio is empty!")
```

```
snp_500_2024_value = snp_500_2024.xs('Close', level='Price', axis=1)
snp_500_2024_value = snp_500_2024_value.loc[:, snp_500_2024_value.columns[0]]  # Use 1st stock as base
snp_500_2024_value = snp_500_2024_value / snp_500_2024_value.iloc[0] * amount
```

```
all_portfolios = pd.DataFrame(index=price_data_2024.index)
all_portfolios["snp_500"] = snp_500_2024_value.values
all_portfolios["central_portfolio"] = central_portfolio_value.values
all_portfolios["peripheral_portfolio"] = peripheral_portfolio_value.values
all_portfolios.head()
```

Date	snp_500	central_portfolio	peripheral_portfolio
2024-07-08	500.000000	500.000000	500.000000
2024-07-09	486.373200	503.727677	503.571138
2024-07-10	492.961926	510.934628	509.178373
2024-07-11	505.241148	512.830684	511.281301
2024-07-12	505.989867	517.485185	515.555686

Date	snp_500	central_portfolio	peripheral_portfolio
2024-07-08	500.000000	500.000000	500.000000
2024-07-09	486.373200	503.727677	503.571138
2024-07-10	492.961926	510.934628	509.178373
2024-07-11	505.241148	512.830684	511.281301
2024-07-12	505.989867	517.485185	515.555688

```
[ ] all_portfolios # returns via date !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Date	snp_500	central_portfolio	peripheral_portfolio
2024-07-08	500.000000	500.000000	500.000000
2024-07-09	486.373200	503.727677	503.571138
2024-07-10	492.961926	510.934628	509.178373
2024-07-11	505.241148	512.830684	511.281301
2024-07-12	505.989867	517.485185	515.555688
...	...	...	...
2024-12-24	491.062323	644.649141	429.837422
2024-12-26	490.308464	645.223251	430.374383
2024-12-27	489.403822	637.743804	427.636389
2024-12-30	484.729945	631.836088	421.991189
2024-12-31	486.539229	631.422901	424.709436

124 rows × 3 columns

Network-Based Stock Portfolio Optimization Report

```python
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(16, 8))
ax.plot(all_portfolios["snp_500"], label='S&P 500')
ax.plot(all_portfolios["central_portfolio"], label='Central Portfolio')
ax.plot(all_portfolios["peripheral_portfolio"], label='Peripheral Portfolio')
ax.legend(loc='upper left')
plt.show()
```

Network-Based Stock Portfolio Optimization Report

Principal 1,000

```
[ ]  amount = 1000  # new principal investment

     # Central Portfolio
     central_portfolio_value = pd.DataFrame()
     for stock in central_portfolio:
         central_portfolio_value[stock] = price_data_2024[stock]

     if not central_portfolio_value.empty:
         portfolio_unit = central_portfolio_value.sum(axis=1).iloc[0]
         share = amount / portfolio_unit
         central_portfolio_value = central_portfolio_value.sum(axis=1) * share
     else:
         print("Central portfolio is empty!")

     # Peripheral Portfolio
     peripheral_portfolio_value = pd.DataFrame()
     for stock in peripheral_portfolio:
         peripheral_portfolio_value[stock] = price_data_2024[stock]

     if not peripheral_portfolio_value.empty:
         portfolio_unit = peripheral_portfolio_value.sum(axis=1).iloc[0]
         share = amount / portfolio_unit
         peripheral_portfolio_value = peripheral_portfolio_value.sum(axis=1) * share
     else:
         print("Peripheral portfolio is empty!")
```

```
[ ]  snp_500_2024_value = snp_500_2024.xs('Close', level='Price', axis=1)
     snp_500_2024_value = snp_500_2024_value.loc[:, snp_500_2024_value.columns[0]]
     snp_500_2024_value = snp_500_2024_value / snp_500_2024_value.iloc[0] * amount

     all_portfolios = pd.DataFrame(index=price_data_2024.index)
     all_portfolios["snp_500"] = snp_500_2024_value.values
     all_portfolios["central_portfolio"] = central_portfolio_value.values
     all_portfolios["peripheral_portfolio"] = peripheral_portfolio_value.values
```

```
[ ]  all_portfolios.head()
```

Date	snp_500	central_portfolio	peripheral_portfolio
2024-07-08	1000.000000	1000.000000	1000.000000
2024-07-09	972.746399	1007.455354	1007.142275
2024-07-10	985.923853	1021.869255	1018.356747
2024-07-11	1010.482296	1025.661368	1022.562601
2024-07-12	1011.979734	1034.970369	1031.111371

Network-Based Stock Portfolio Optimization Report

```
all_portfolios.head()
```

Date	snp_500	central_portfolio	peripheral_portfolio
2024-07-08	1000.000000	1000.000000	1000.000000
2024-07-09	972.746399	1007.455354	1007.142275
2024-07-10	985.923853	1021.869255	1018.356747
2024-07-11	1010.482296	1025.661368	1022.562601
2024-07-12	1011.979734	1034.970369	1031.111371

```
[ ] all_portfolios
```

Date	snp_500	central_portfolio	peripheral_portfolio
2024-07-08	1000.000000	1000.000000	1000.000000
2024-07-09	972.746399	1007.455354	1007.142275
2024-07-10	985.923853	1021.869255	1018.356747
2024-07-11	1010.482296	1025.661368	1022.562601
2024-07-12	1011.979734	1034.970369	1031.111371
...	...	...	...
2024-12-24	982.124646	1289.298283	859.674844
2024-12-26	980.616928	1290.446501	860.748767
2024-12-27	978.807644	1275.487608	855.272778
2024-12-30	969.459890	1263.672176	843.982379
2024-12-31	973.078457	1262.845802	849.418871

124 rows × 3 columns

Network-Based Stock Portfolio Optimization Report

```
[ ] import matplotlib.pyplot as plt

    fig, ax = plt.subplots(figsize=(16, 8))
    ax.plot(all_portfolios["snp_500"], label='S&P 500')
    ax.plot(all_portfolios["central_portfolio"], label='Central Portfolio')
    ax.plot(all_portfolios["peripheral_portfolio"], label='Peripheral Portfolio')
    ax.legend(loc='upper left')
    plt.show()
```