UG Project

Testing the efficiency of financial markets using Mantegna Asset Trees and Asset Graphs

Ayush Gupta, Roll No: 20124010
IDD - 6th Semester
Under the guidance of Dr. Subir Das

OVERVIEW

- Study of clustering of NIFTY 50 companies using the correlation matrix of asset returns
- The correlations are transformed into distances between stocks, whose subset is selected using the minimum spanning tree criterion
- The tree is then further utilized to identify most dominate stocks and industries in the Indian financial markets to assist decisions

Motivation

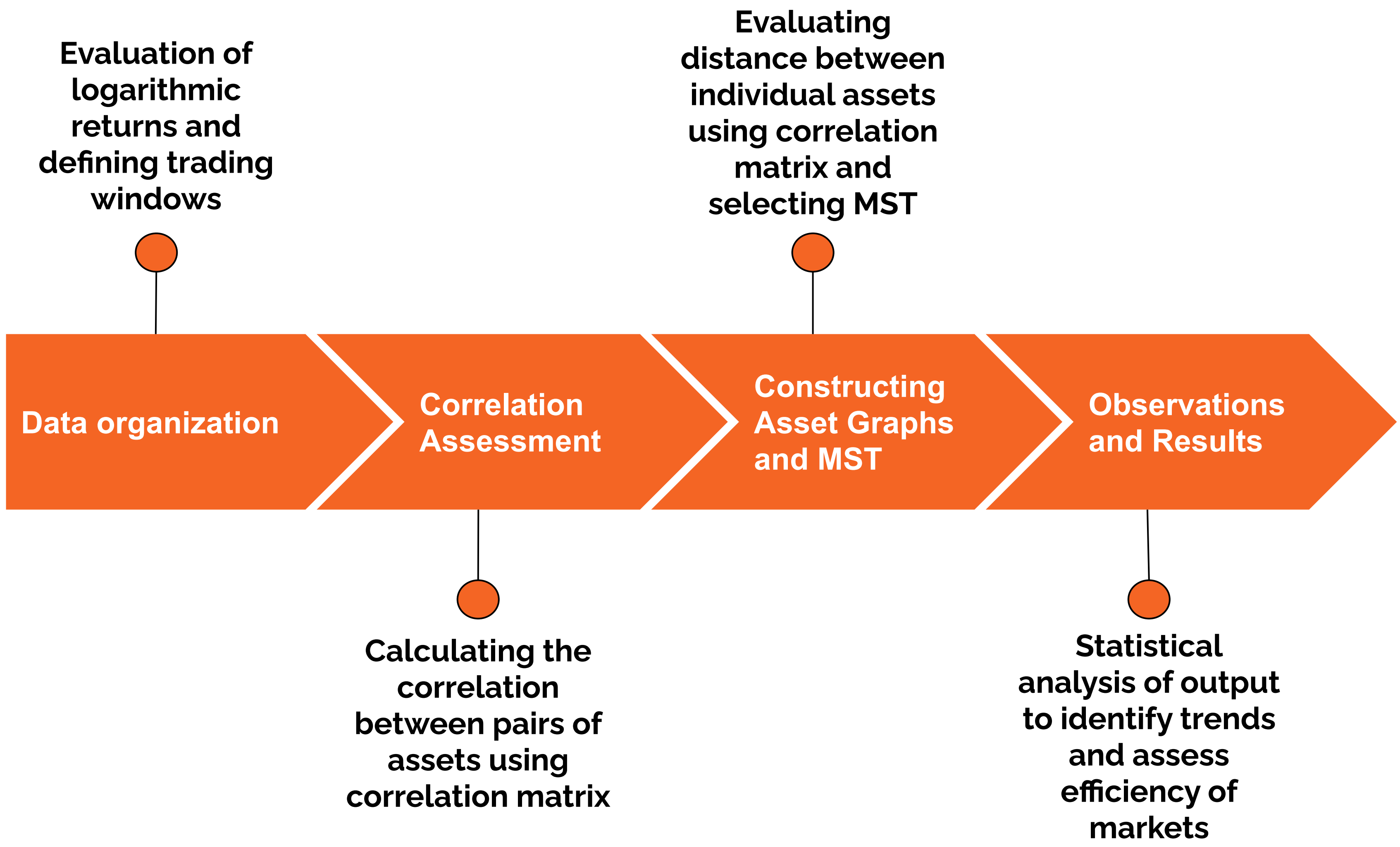
As algorithmic trading softwares takeover the financial markets, it becomes interesting to analyze how "efficient" our markets are.

An "efficient" asset market is one in which in the information contained in the past prices is instantly, fully, and continually reflected in the asset's current price. Hence, the more efficient the market is, the more random is the sequence of price changes generated by the market.

We try to assess Markowitz Portfolio Optimization Theory using asset trees and asset graphs developed by Mantegna

The following report encompasses the methodology and results derived from this study on the 20 years performance of the Indian financial market using top 30 NIFTY50 stocks

Project Report Structure


Evaluation of logarithmic returns and defining trading windows
Evaluating distance between individual assets using correlation matrix and selecting MST
Data organization
Correlation Assessment
Constructing Asset Graphs and MST
Observations and Results
Calculating the correlation between pairs of assets using correlation matrix
Statistical analysis of output to identify trends and assess efficiency of markets

Data Organization

Data collection and organization

For my research, I collected data of closing prices of **N = 30 stocks in NIFTY 50** which are most dominant in the portfolio mix for the past 20 years [21 December 2002 - 23 December 2022], amounting to a total of **4972 price quotes per stock**, which are then indexed by a time variable $\tau = 0, 1, 2, 3, \ldots, 4971$.

Stock price is highly erratic in nature, hence, we take up a **smoothing process** to ease the analysis by **dividing the price quotes into time windows**. We create M windows, $t = 0, 1, 2, 3, \ldots, M - 1$ of width T, where **T represents the number of daily returns included in the window**.

Several consecutive windows overlap with each other whose extent is measured using **window step length parameter δT**, which describing the displacement between two adjacent trading windows, and it is also measured in terms of number of trading days.

If the window size is too small, the output will have several outliers which make the overall dataset noisy. At the same time, a window too large exceedingly smoothens the data, resulting in loss of critical data points. Hence, **the trading window size is a trade-off between noise and smoothness.**

Evaluation of returns

The optimal results are obtained from monthly stepped four-year windows, i.e.
Number of working days in a year ~ 250 days
T = 1000 days
δT = 250 / 12 = ~21 days
And hence, M = 190 windows

In order to investigate correlations between stocks we first denote the closure price of stock i at time τ by $P_i(\tau)$. To evaluate the relative returns, we shall be utilizing the logarithmic returns of stock, given by

$$r(t) = \ln P(t) - \ln P(t-1)$$

This process is completed for all the 30 stocks, until we have a 30 x 4971 matrix representing the relative returns across all assets.

The **sequence of such returns** contained in a given trading window t, forms **the return vector r_i^t.**

Storing and segmenting stocks

[42]
```python
stocks = os.listdir("/content/drive/MyDrive/Classroom/StreamProject")

nifty50 = "/content/drive/MyDrive/Classroom/StreamProject/"
print(stocks)
print(len(stocks))
```

```
['ADANIENT.NS.csv', 'INDUSINDBK.NS.csv', 'AXISBANK.NS.csv', 'ICICIBANK.NS.csv', 'ULTRACEMCO.NS.csv', 'N
30
```

[43]
```python
# Industries

banking_and_finance = ['INDUSINDBK.NS', 'AXISBANK.NS', 'ICICIBANK.NS', 'HDFCBANK.NS', 'HDFC.NS']
industries = ['GRASIM.NS', 'ULTRACEMCO.NS', 'HINDALCO.NS', 'ASIANPAINT.NS', 'TATASTEEL.NS', ]
auto = ['M&M.NS', 'BAJAJ-AUTO.NS', 'HEROMOTOCO.NS', 'EICHERMOT.NS']
consumer_goods = ['NESTLEIND.NS', 'HINDUNILVR.NS', 'TITAN.NS', 'BRITANNIA.NS', 'ITC.NS']
software = ['HCLTECH.NS', 'WIPRO.NS', 'INFY.NS', 'LT.NS']
power_and_energy = ['ONGC.NS', 'RELIANCE.NS', 'BPCL.NS']
health_and_pharma = ['SUNPHARMA.NS', 'CIPLA.NS', 'APOLLOHOSP.NS']
misc = ['ADANIENT.NS']

sectors = [banking_and_finance, industries, auto, consumer_goods, software, power_and_energy, hea
sector_names = ["banking_and_finance", "industries", "auto", "consumer_goods", "software", "power_
```

Creating arrays of closing price

[46]
```python
# Finding the value of first  using the path of the file

path = nifty50 + stocks[0]
myStr = idx[0]

df = pd.read_csv(filepath_or_buffer = path, index_col = "Date")
df = df.Close
df = df.fillna(method = "ffill")
arr = df.to_numpy()
# arr = np.round(arr, 2)
arr = np.expand_dims(arr, axis = 0) # Expanding dimensions from 1 to 2

# Converting it to variable s0

vars().__setitem__(myStr, arr)

close = s0 # Array that will contain all the closing prices

for file in range(1, len(stocks)): # Parsing the length of the directory
    path = nifty50 + stocks[file]
    myStr = idx[file]

    df = pd.read_csv(filepath_or_buffer = path, index_col = "Date")
    df = df.Close
    df = df.fillna(method = "ffill")
    arr = df.to_numpy()
    arr = np.round(arr, 2)
    arr = np.expand_dims(arr, axis = 0)
    close = np.concatenate((close, arr), axis = 0)

    vars().__setitem__(myStr, arr)


print(close.shape)
```

```
(30, 4972)
```

Calculating the relative returns of the individual stocks to form a 30x4971 matrix

```
returns = np.empty(close.shape) # Creating an empty array to store returns

for i in range(close.shape[0]): # Using a loop function to find the relative change in stock price using r(t) = ln{P(t)} - ln{P(t - 1)}
    for j in range(1, close.shape[1]):
        returns[i][j] = math.log(close[i][j]) - math.log(close[i][j - 1])

rel_returns = np.delete(returns, 0, axis = 1) # Removing first column for j = 0

rel_returns[np.isnan(rel_returns)] = 0

print(rel_returns)
```

```
[[-0.01236212  0.          0.00757307 ... -0.01568966  0.02346156
   0.02184388]
 [ 0.02454111  0.         -0.01526747 ... -0.0081405  -0.00489178
   0.00415954]
 [ 0.00806921  0.         -0.00345026 ... -0.00325638  0.01180049
   0.00374483]
 ...
 [ 0.00644948  0.          0.03034836 ... -0.00495729  0.01304981
   0.00861893]
 [ 0.00170358  0.          0.01212478 ... -0.02142215 -0.00131762
  -0.00308121]
 [-0.00744504  0.         -0.00067958 ... -0.01112854  0.01598613
  -0.00102843]]
```

Correlation Assessment

Calculation of time correlation coefficients

To characterize the synchronous time evolution of assets, we use equal time correlation coefficients between assets i and j as,

$$\rho_{ij}^{t} = \frac{\langle \boldsymbol{r}_i^t \boldsymbol{r}_j^t \rangle - \langle \boldsymbol{r}_i^t \rangle \langle \boldsymbol{r}_j^t \rangle}{\sqrt{[\langle {\boldsymbol{r}_i^t}^2 \rangle - \langle \boldsymbol{r}_i^t \rangle^2][\langle {\boldsymbol{r}_j^t}^2 \rangle - \langle \boldsymbol{r}_j^t \rangle^2]}}$$

where <...> represents the time average over the consecutive trading days in the return vectors.

The coefficients of correlation fulfill the condition **$-1 \le \rho_{ij} \le 1$**.
If $\rho_{ij} = 1$, then the stock prices are completely correlated
If $\rho_{ij} = 0$, then the stock prices are uncorrelated (which should be the case for efficient markets)
If $\rho_{ij} = -1$, then the stock prices are completely anti-correlated

These correlation coefficients form an **N x N symmetric correlation matrix C^t** which serves as the basis for the formation of trees to further analyze the correlation between stocks

[51]

```python
# Empty correlation symmetric matrix with dim0 as time interval

corr = np.empty((195, 30, 30))

for i in range(195):
    for j in range(30):
        corr[i][j][j] = 1 # A stock index is perfectly correlated to itself
```

```python
def correlation_arr(a, corr, t):
    n = a.shape[0]

    for i in range(n):
        for j in range(i + 1, n):
            corr[t][i][j] = correl(a[i], a[j])
            corr[t][j][i] = corr[t][i][j]

    return corr
```

[53]

```python
# Creating the time windows t using T and delT

t = 0

for i in range(0, rel_returns.shape[1], delT):
    # Shape of close = (30, 4972)

    if(i + T > rel_returns.shape[1]):
        break

    windows = rel_returns[:, i:i+T]

    correlation_arr(windows, corr, t)

    t += 1
```

```python
def roll_avg(a):  # Function to calculate the average of a
    return np.mean(a)

def correl(a, b):  # Function to find the correlation betwe
    ab = a * b
    a2 = a ** 2
    b2 = b ** 2

    num = roll_avg(ab) - (roll_avg(a) * roll_avg(b))
    den = math.sqrt((roll_avg(a2) - (roll_avg(a) ** 2))
    * (roll_avg(b2) - (roll_avg(b) ** 2)))

    cor = num / den

    return cor
```

Evaluation formula for correlation coefficients

Preparation of arrays to store correlation coefficients

Constructing Asset Graphs and MST

Construction of Asset Graph

In order to construct an asset graph, and subsequently an asset tree, we start by defining the assets as the nodes, and the "distance" between the relative returns of the assets as the edges of a graph. We do so by defining the distance as,

$$\boxed{d_{ij}^{t} = \sqrt{(2 * (1 - \rho_{ij}^{t}))}}$$

Such obtained distances satisfy the condition $\mathbf{0 \leq d_{ij} \leq 2}$.
If $d_{ij} = 0$, then the stock prices are completely correlated
If $d_{ij} = 1$, then the stock prices are uncorrelated (which should be the case for efficient markets)
If $d_{ij} = 2$, then the stock prices are completely anti-correlated

Thus we have obtained an **N x N symmetric distance matrix $\mathbf{D^t}$** which represents the **asset graph** with the following properties:-

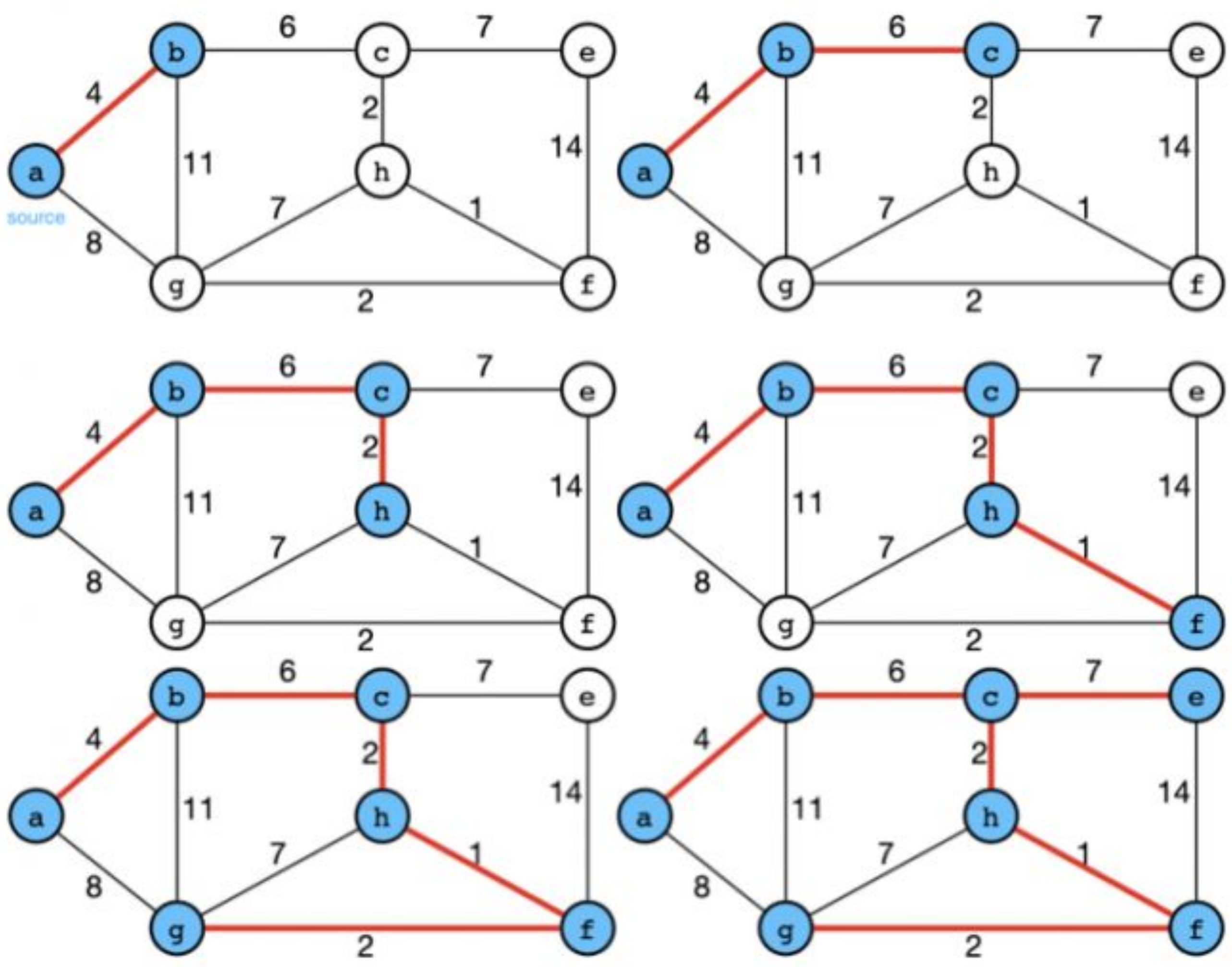
- It is a completely connected graph
- It contained 30 vertices and 435 edges
- The term d_{ij} represents the length (weight) of the edge connecting asset i and j

Mantegna's Asset Trees

In order to construct an asset tree, we need to first understand the that the approach requires a hypothesis about the topology of the metric space, namely, the **ultrametricity hypothesis.** In practice, it leads to determining the **minimum spanning tree (MinST)** of the distances, denoted **T^t**. The spanning tree is a simply connected acyclic (no cycles) graph that connects **all N nodes (stocks) with N −1 edges** such that the **sum of all edge weights $\Sigma_{d_{ij} \epsilon T^t} d_{ij}$ is minimum.**

We refer to the minimum spanning tree at time t by **$T^t = (V, E^t)$**, where V is the corresponding set of vertices (or assets) and E^t is the corresponding set of unordered edges. Since the stocks remain the same as time passes, **V is independent of time.** However, E^t may change as the weight of edges changes, because D^t shall evolve with time. Hence, **E^t is time dependent.**

Therefore, the MinST represents the strongest correlations connecting all the stocks. We define the centre of such an MinST as the vertex with the highest degree: the stock is the greatest number of strong correlation with other stocks.


a
b
c
e
h
g
f
source
4
6
7
2
11
14
7
1
8
2

```
dist = np.empty(rho.shape)

dist = (2 * (1 - rho)) ** (1/2) # Calculating the distance between two stock indices
print(dist.shape)

# dist is the symmetric distance matrix between the 30 stocks
```

```
(190, 30, 30)
```

```
graphs = [] # Defining a list to store all of the graphs

for i in range(dist.shape[0]):
    graph = nx.from_numpy_array(dist[i, :, :]) # Creating a graph from the distance array
    graphs.append(graph) # Appending the graph to the list of graphs

print(graphs[0])
```

```
Graph with 30 nodes and 435 edges
```

```
center = [] # Collecting the central vertex (stock) in a list

stocks = [sub[:-4] for sub in stocks]

for i in range(len(T)):
    ctr = np.zeros(30)
    e = sorted(T[i].edges(data = False))

    for j in range(len(e)):
        ctr[e[j][0]] += 1
        ctr[e[j][1]] += 1

    index = getMaxValue(ctr, max(ctr))

    center.append(stocks[index])
```

```
# Dominant stocks in the market sorted in descending order of dominance

dom_stocks = sorted(center, key = center.count, reverse = True)
```

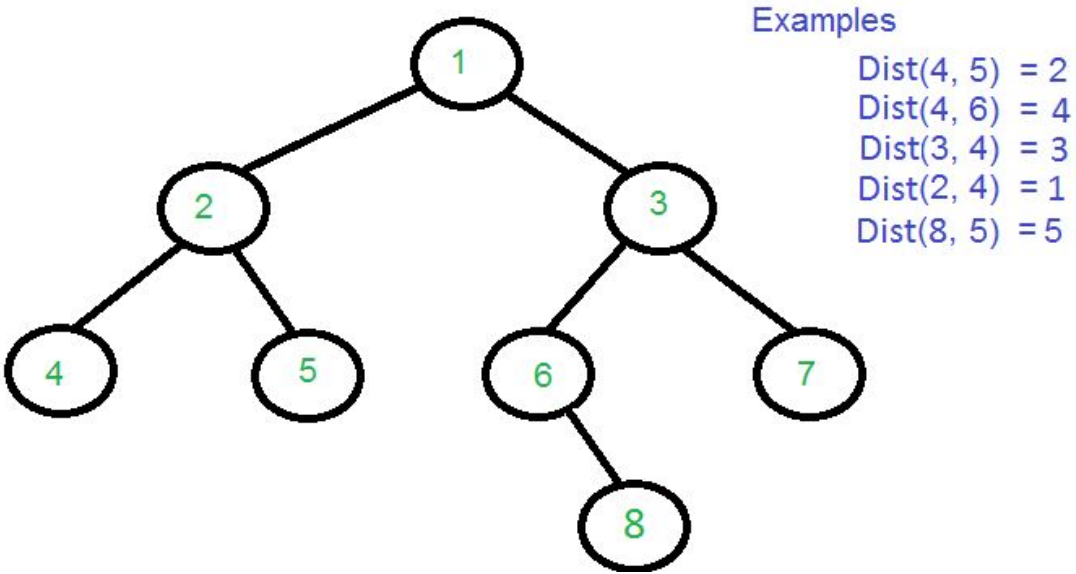
Minimum Spanning Tree

```
T = [] # List to store all spanning trees

for i in range(len(graphs)):
    spanning_tree = nx.minimum_spanning_tree(G = graphs[i], algorithm = 'prim', ignore_nan = True) # Spanning Tree for ith graph
    T.append(spanning_tree) # Appending the spanning tree to the list of spanning trees

print(T[0])
```

```
Graph with 30 nodes and 29 edges
```

“Distance” between assets

Clearly, a tree shall consist of N nodes and N-1 edges, which translates to a finite number of edges between any two numbered nodes, with the number of edges labelled as *lev* being ≥ 1, which can be calculated using the shortest distance path between two nodes.



```
[ ]
# Evaluating distance of the tree nodes from the root node for all 190 MSTs
distances = []

for t in range(len(T)):
    shortest_path_lengths = nx.shortest_path_length(T[t], roots[t])
    distances.append(shortest_path_lengths)

print(len(distances))

# Converting the distances dictionary into an array with the index of the array representing the node
level = np.empty((190, 30))

for i in range(len(distances)):
    for j in range(len(stocks)):
        level[i][j] = distances[i][j]

print(level[3])
```

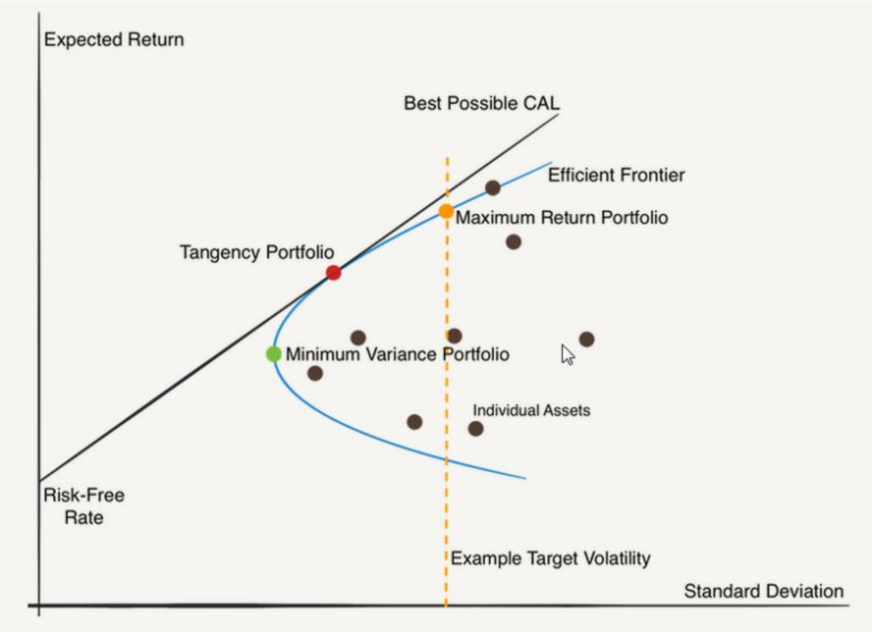
```
190
[6. 3. 2. 2. 3. 1. 3. 2. 4. 1. 3. 2. 0. 1. 2. 1. 4. 6. 4. 1. 6. 2. 6. 2.
 1. 5. 4. 6. 1. 4.]
```

Markowitz Portfolio Optimization Theory

Markowitz model is a method of **maximizing the returns within a calculated risk**. It mainly focuses on portfolio diversification to separate stocks on the basis of risk associated with the asset.

The **efficient frontier parabola** is a graphical representation of the Markowitz model. It shows the optimal portfolios that provide the highest expected return for a given level of risk. The efficient frontier is the boundary of the set of feasible portfolios that maximize expected return for a given level of risk.

Minimum Variance: Marks change from convex to concave
Tangency Portfolio: Most optimal - highest Sharpe ratio
Maximum Return: Highest volatility and returns



Markowitz Portfolio Optimization in action

To implement Markowitz Portfolio Optimization, we need to specify the **expected returns** and covariance matrix of the assets in the portfolio. The expected returns are estimated using the available data, while the covariance matrix measures the degree of correlation between the assets.

The expected returns are evaluated using the percentage change in closing price of stocks over the time window of 1000 days (4 years) with steps of 21 days (1 month) [Since there 21 working days in a month]

$$\% \text{ change} = \frac{x_2 - x_1}{x_1}$$

After getting the expected returns for each asset along with the covariance matrix, **we define our target returns using θ - the affinity towards taking risk**, where r_m is the risk associated with least exposed asset and r_M is the risk associated with the most exposed asset.

$$r_{\mathbf{P},\theta} = (1-\theta) r_m + \theta r_M$$

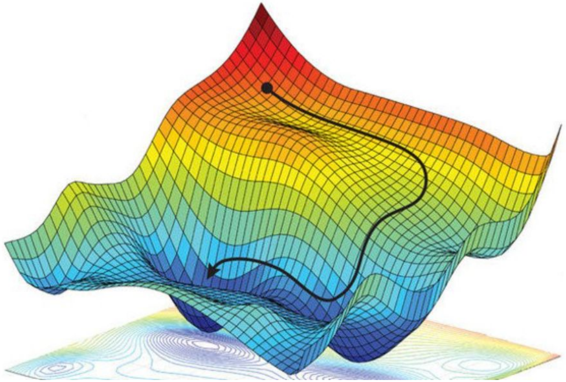
Once we have estimated the expected returns and covariance matrix, we can use a solver to find the optimal portfolio. The solver will find the set of weights for each asset that maximizes the expected return for a given level of risk.

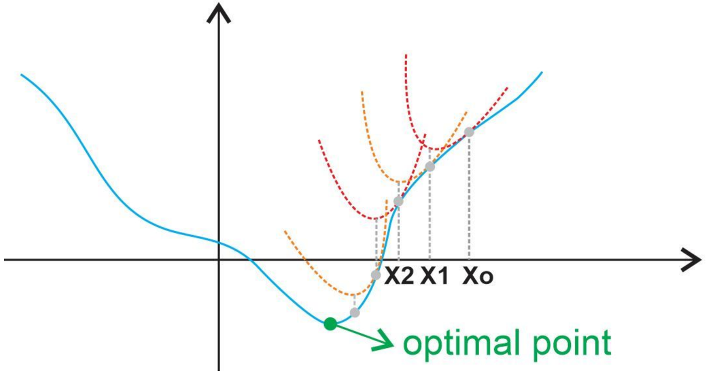
Convex Optimization

The optimization algorithm typically involves solving a **quadratic programming problem**. The objective function of the problem is to **maximize the expected return** of the portfolio, **subject to a set of constraints on the portfolio weights**.

The constraints on the portfolio weights ensure that the weights add up to 1.0, and that each weight be between 0 and 1. The covariance matrix is used to model the correlations between the assets in the portfolio, which affects the level of risk and return of the portfolio.

The solver algorithm solves the optimization problem by iterating through different sets of portfolio weights, evaluating the objective function at each set of weights, and adjusting the weights until the optimal solution is found. This process is repeated until the algorithm converges to a set of weights that satisfy the constraints and maximize the expected return.





```python
import cvxpy as cp

weights = np.empty((190, 30))
weights_low = np.empty((190, 30))
weights_high = np.empty((190, 30))

theta = 0.1
target_low = (1-theta) * min_return + theta * max_return
theta = 0.26
target_avg = (1-theta) * min_return + theta * max_return
theta = 0.6
target_high = (1-theta) * min_return + theta * max_return

for i in range(len(distances)):
    cov = rho[i]
    mu = expected_ret[:, i]
    x = cp.Variable(30)
    objective = cp.Minimize(cp.quad_form(x, cov))

    constraints_low = [
        x >= 0,
        cp.sum(x) == 1,
        cp.sum(mu.T @ x) >= target_low,
        cp.sum(mu.T @ x) <= target_avg
    ]
    constraints_avg = [
        x >= 0,
        cp.sum(x) == 1,
        cp.sum(mu.T @ x) >= target_avg
    ]

    constraints_high = [
        x >= 0,
        cp.sum(x) == 1,
        cp.sum(mu.T @ x) >= target_high
    ]

    problem = cp.Problem(objective, constraints_avg)
    problem.solve()
    weights[i] = x.value

    problem = cp.Problem(objective, constraints_low)
    problem.solve()
    weights_low[i] = x.value

    problem = cp.Problem(objective, constraints_high)
    problem.solve()
    weights_high[i] = x.value
```

```python
print(weights[150])
print(weights_low[150])
print(weights_high[150])
```

```
[ 1.32460920e-02  1.67636816e-02  3.48664544e-02  5.70177139e-03
 -4.37391197e-23  8.83789519e-02  3.61596235e-02  3.39786371e-02
  9.48715073e-02  9.40610488e-02  1.00884586e-02  1.05548529e-22
  7.45089858e-02  6.12262381e-02  4.54626563e-02  1.15057192e-02
  8.44191700e-03 -4.22995192e-23  6.22323282e-02  5.02465455e-23
  2.14160831e-02 -9.30382278e-23  3.03938632e-02  7.71711238e-02
  1.08030397e-22  1.67955117e-03  2.11995696e-02  7.89041396e-02
  5.25644508e-02  2.51771476e-02]
```

Computing mean layer of portfolio

The weighted portfolio layer is found out by defining the mean layer for a specific portfolio P and risk affinity θ in the following manner

$$l_{\mathbf{P}}(t,\theta) = \sum_{i \in \mathbf{P}(t,\theta)} w_i \operatorname{lev}(v_i^t)$$

$$\text{where } \textstyle\sum_{i=1}^{N} w_i = 1$$

$$w_i \geq 0 \text{ for all } i$$

The condition $w_i \geq 0$ is equivalent to assuming that there is no short-selling. The purpose of this constraint is to prevent negative values for $\ell_P(t)$, which would not have a meaningful interpretation in our framework of dynamic trees with central vertex.

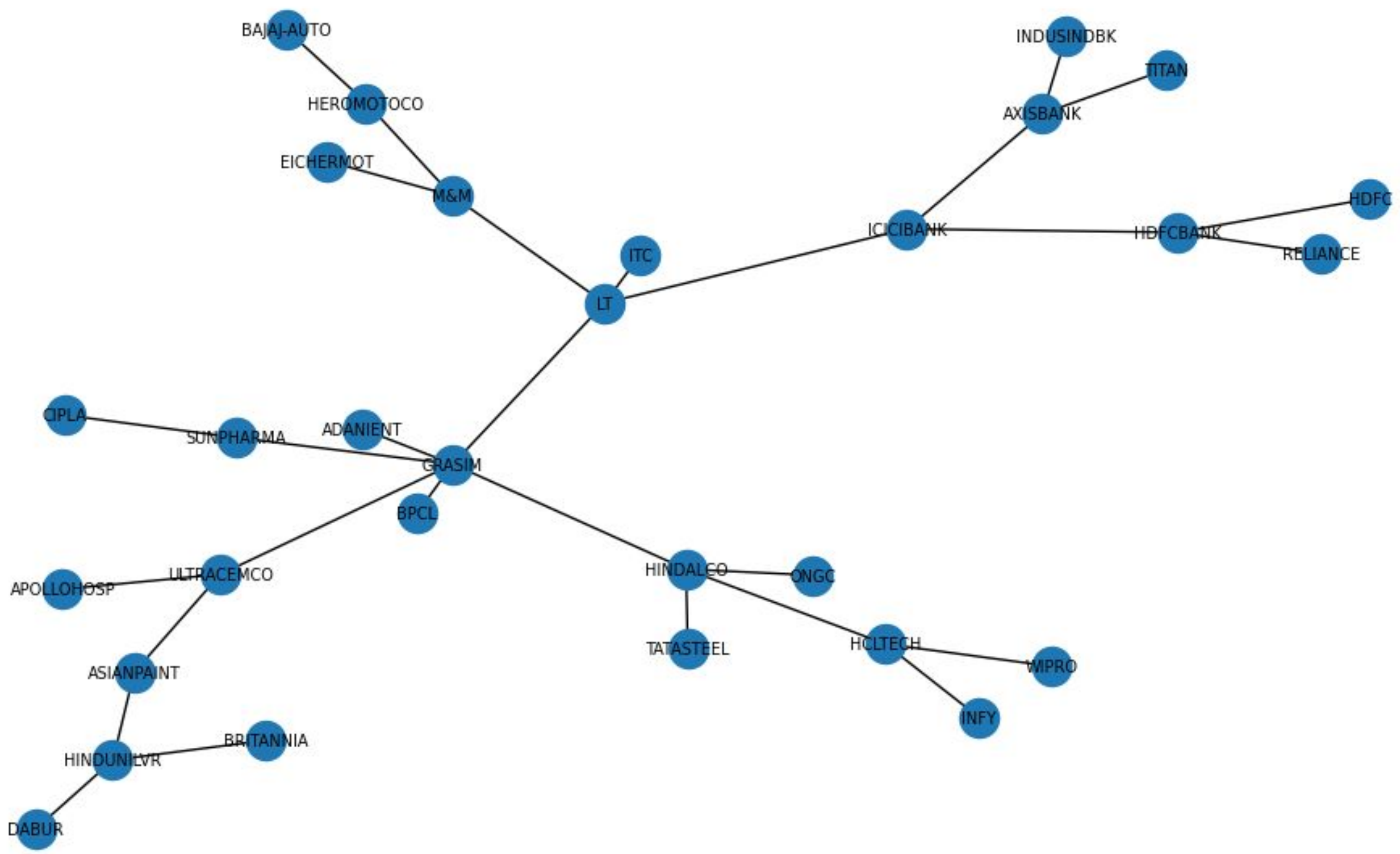
```
# Computing the weighted portfolio layer
wt_layer = np.empty(190)
wt_layer_low = np.empty(190)
wt_layer_high = np.empty(190)

for i in range(len(distances)):
    wt_layer[i] = weights[i] @ level[i].T
    wt_layer_low[i] = weights_low[i] @ level[i].T
    wt_layer_high[i] = weights_high[i] @ level[i].T

print(np.nanmean(wt_layer_low))
print(np.nanmean(wt_layer_high))
```

```
3.080685956550603
2.977829160477921
```

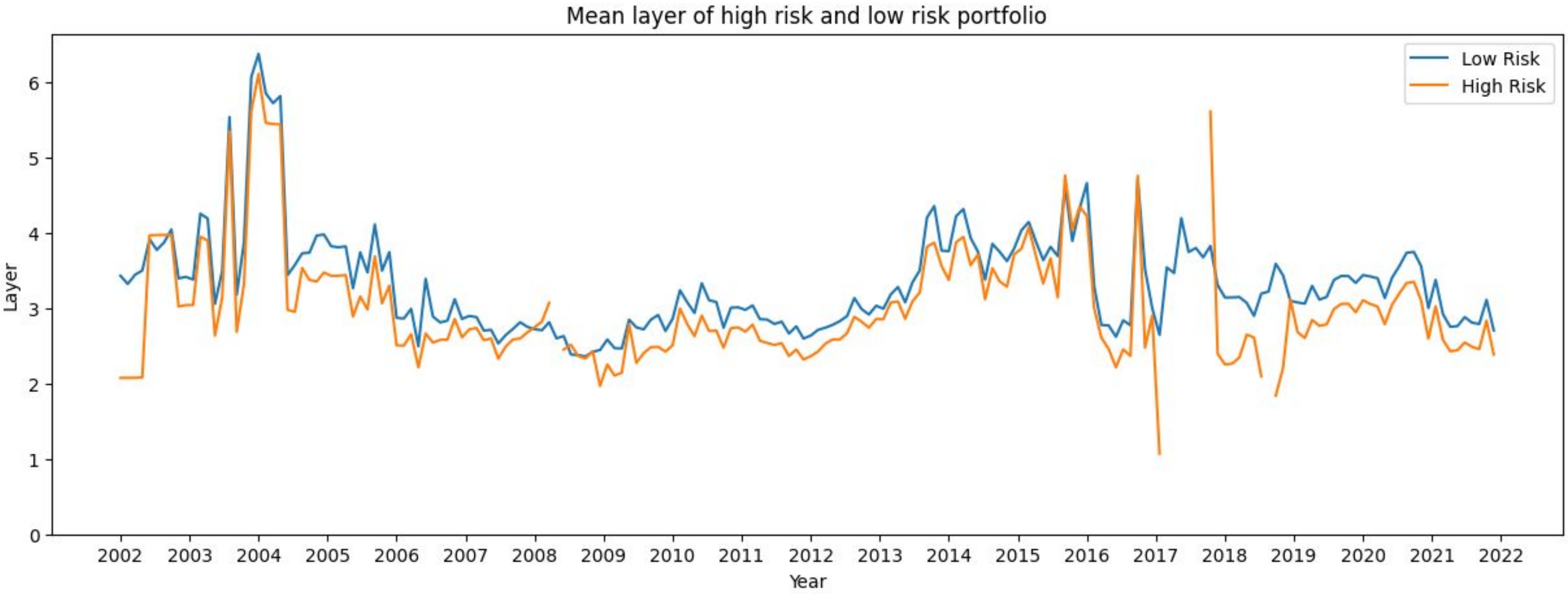
Observations and Results

```python
graphs_nodes = {}

for i in range(len(stocks)):
    graphs_nodes[i] = stocks[i][:-3]

for i in range(len(T)):
    nx.set_node_attributes(T[i], graphs_nodes, 'label')
```

```python
fig = plt.figure(figsize = (10, 6))
tree = T[189]
print(center[189])
pos = nx.spring_layout(tree)
nx.draw(tree, pos)
nx.draw_networkx_labels(tree, pos, graphs_nodes, font_size=7, font_color='black')
plt.show()
```

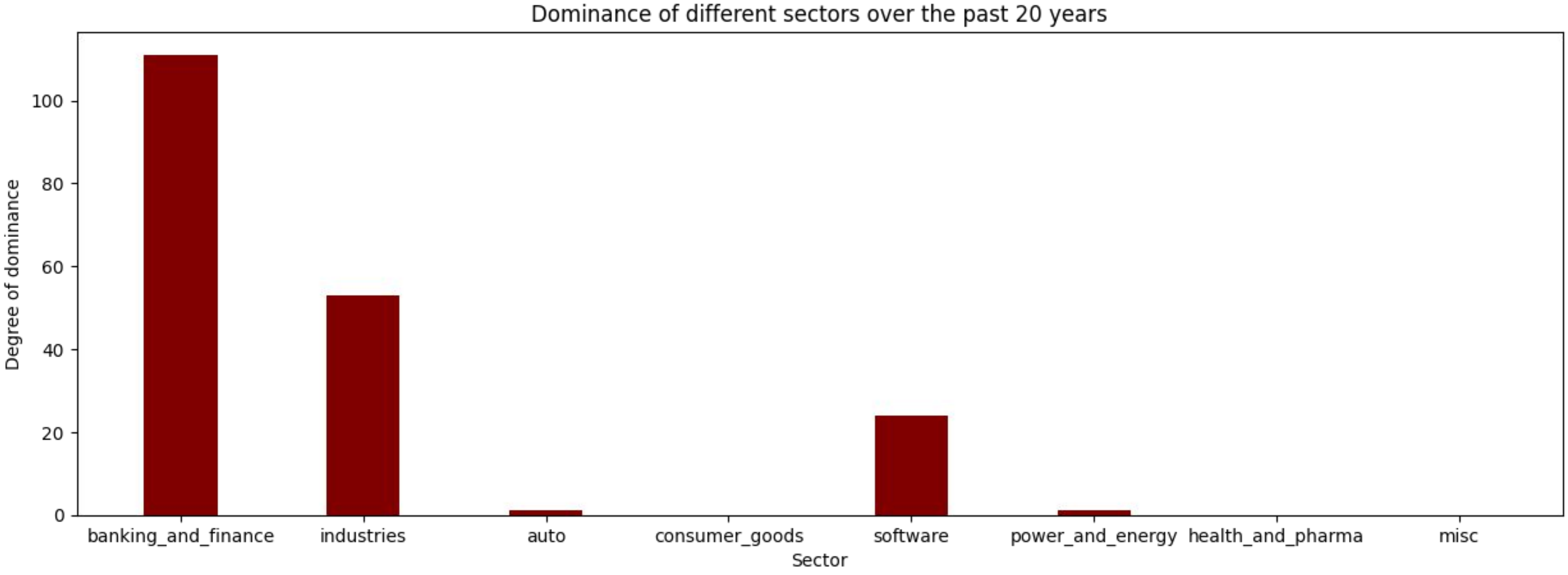

BAJAJ-AUTO
HEROMOTOCO
EICHERMOT
M&M
INDUSINDBK
TITAN
AXISBANK
HDFC
ICICIBANK
HDFCBANK
RELIANCE
ITC
LT
CIPLA
SUNPHARMA
ADANIENT
GRASIM
BPCL
ULTRACEMCO
APOLLOHOSP
HINDALCO
ONGC
TATASTEEL
HCLTECH
WIPRO
INFY
ASIANPAINT
HINDUNILVR
BRITANNIA
DABUR

```python
fig = plt.figure(figsize = (15, 5))

#x = range(190)
#y = wt_layer
#plt.plot(x, y)

x = range(190)
y = wt_layer_low
plt.plot(x, y, label = "Low")

x = range(190)
y = wt_layer_high
plt.plot(x, y, label = "High")

xticks_per_year = 190/20
xticks = [i * xticks_per_year for i in range(21)]
plt.xticks(xticks, range(2002, 2023, 1))
plt.ylim(0)

plt.xlabel("Year")
plt.ylabel("Layer")

plt.legend()

plt.show()
```



Mean Layer: High Risk = 2.97 Low Risk = 3.08

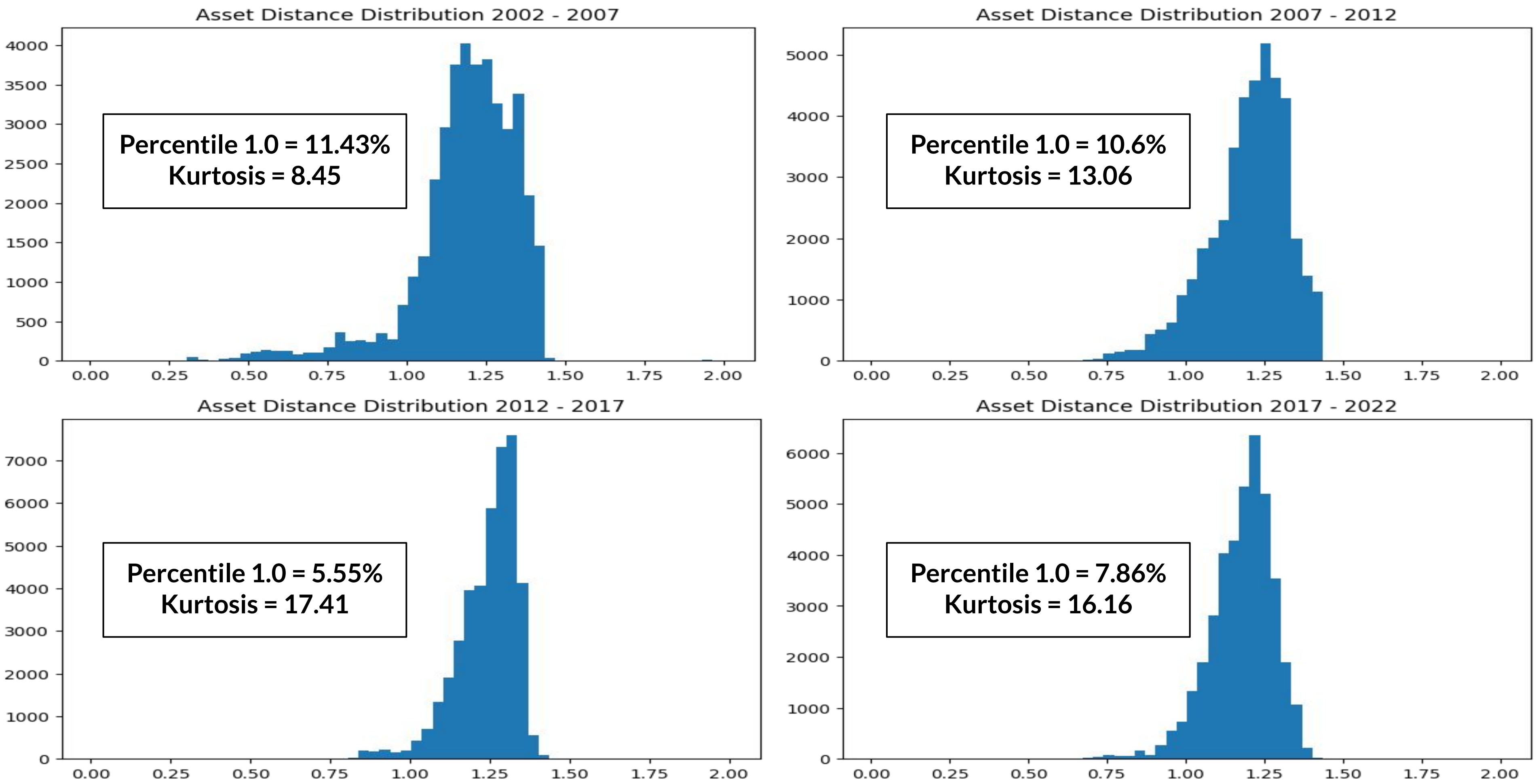
Banking and finance assets have tended to dominate the stock market with the strongest impact on the stock market as a whole



Statistical Analysis

```
from scipy.stats import kurtosis, percentileofscore

def func(ctr):
    if ctr == 0:
        return "02 - 2007"
    if ctr == 1:
        return "07 - 2012"
    if ctr == 2:
        return "12 - 2017"
    if ctr == 3:
        return "17 - 2022"


%matplotlib inline
plt.rcParams.update({'figure.figsize':(7,5), 'figure.dpi':100})

ctr = 0

for i in range(0, dist.shape[0], 48):
    plt.hist(dist[i:i+48, :, :].flatten(), bins=60, range = (0.01, 2.0))

    plt.gca().set(title='Asset Distance Distribution 20' + func(ctr));

    ctr += 1
    plt.show()

    print("Percentile of 1.0 = " + str(np.around(percentileofscore(dist[i:i+48, :, :].flatten(), 1.0), 2)))
    print("Median = " + str(np.around(np.median(dist[i:i+48, :, :]), 2)))
    print("Mean = " + str(np.around(np.mean(dist[i:i+48, :, :]), 2)))
    print("Kurtosis = " + str(np.around(kurtosis(dist[i:i+48, :, :].flatten()), 2)))
```

Anti-correlation between stocks tending to increase with time with rising Kurtosis



CONCLUSION

- Banking and finance assets, followed by industries, are most highly correlated with the financial market
- The distance of the assets from the central vertex of the MinST inversely correlates with the risk associated with the asset
- Larger the asset tree, greater is the portfolio diversification and risk minimization
- Hence, Mantegna's asset trees serve as an excellent visualization tool to assess and compare portfolios during mathematical risk modelling

Thank You!

You may read the code by clicking here