Financial Market Regime Analysis

AI For Investment Management

Professor: Dr. Pavel Paramonov

Team 4

 Alaa AlHallak

 Pablo Diaz

 Davide Olivieri

 Salil Singhal

Takumi Someya

Content:

1 Market Overview

2 Regime Analysis

3 Portfolio creation and Testing

4 Findings and Recommendations

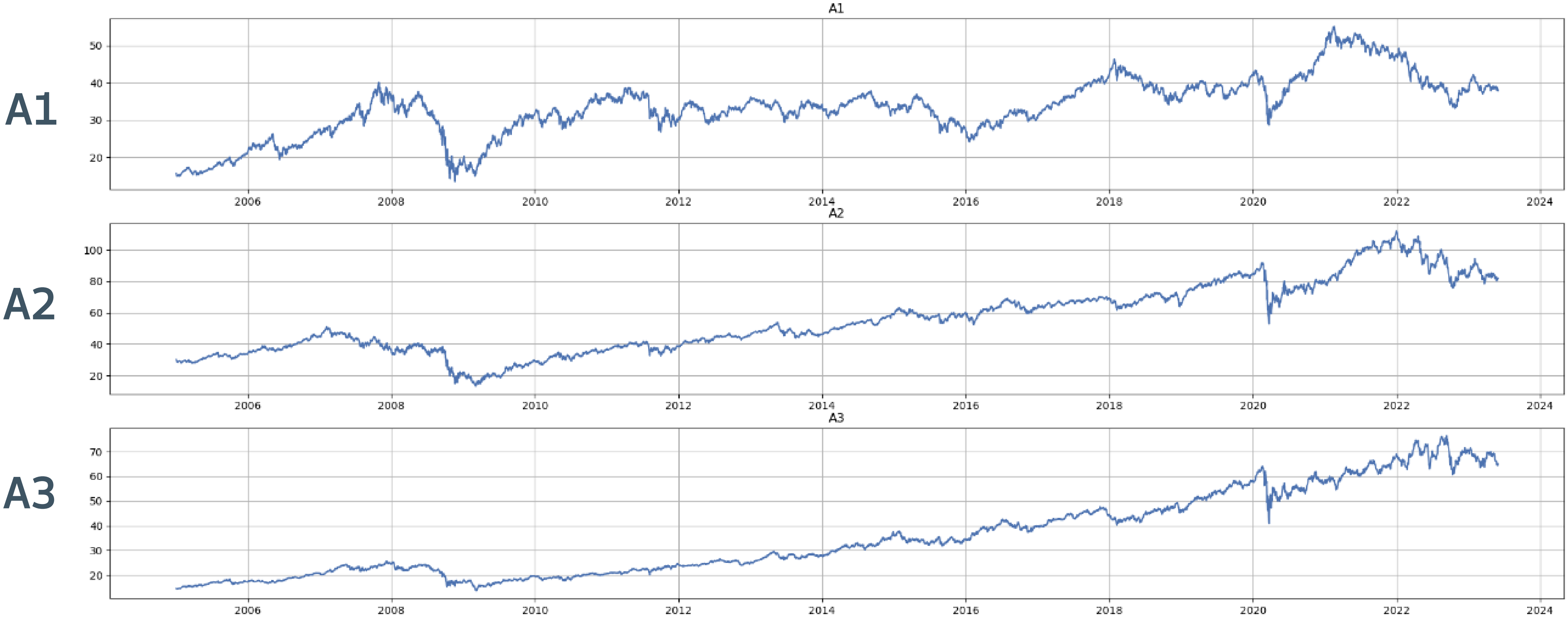
Market Behaviour

Visualization to compare price movement over time for each asset

2007- Financial Crisis

2020-Covid-19 Pandemic

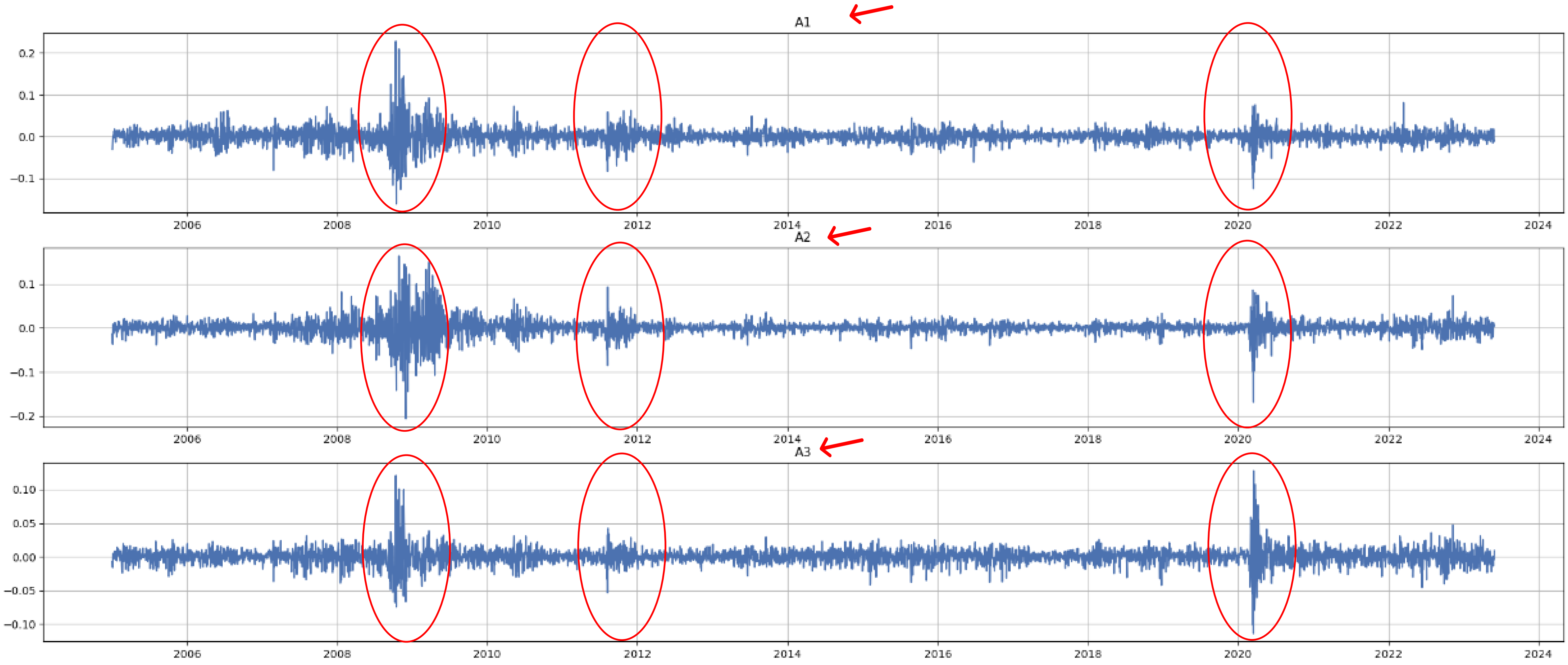
The share price of Assets 2005-2023



Market Behaviour

Daily return patterns of three different assets in separate subplots, allowing for a comparative analysis of their performance over time.

Daily returns of Assets 2005-2023

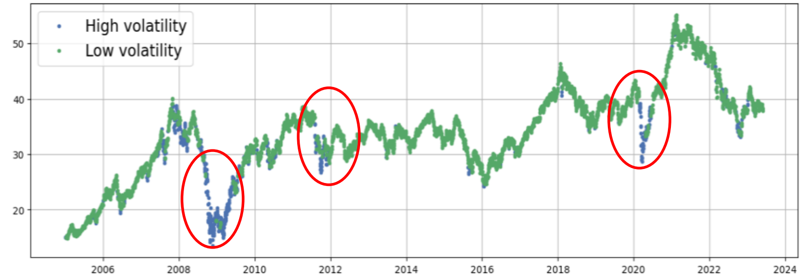


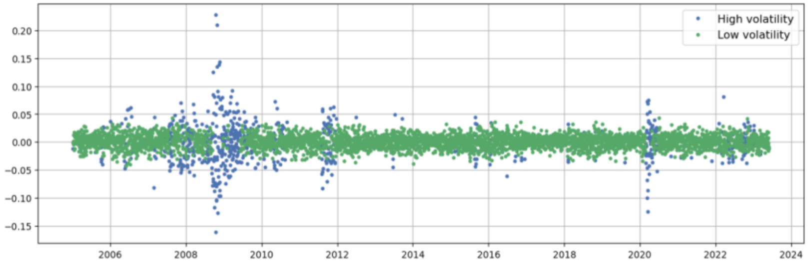
Hidden states of Markov Model

HULT
INTERNATIONAL
BUSINESS SCHOOL

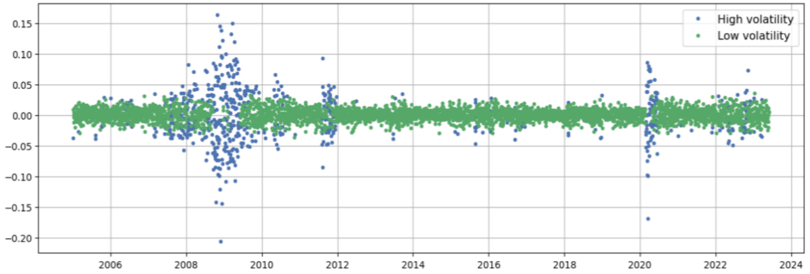
High-volatility and Low-volatility are the two states identified, with A3 following a more consistent upward trend

A1

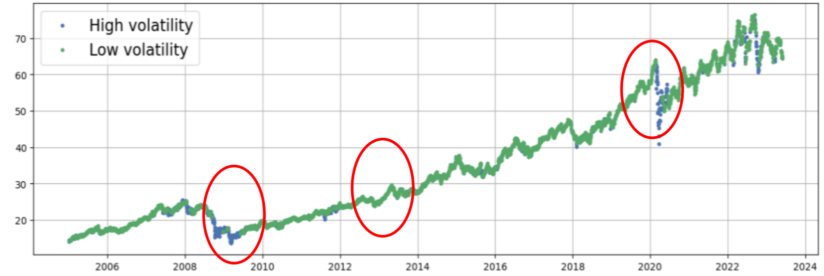


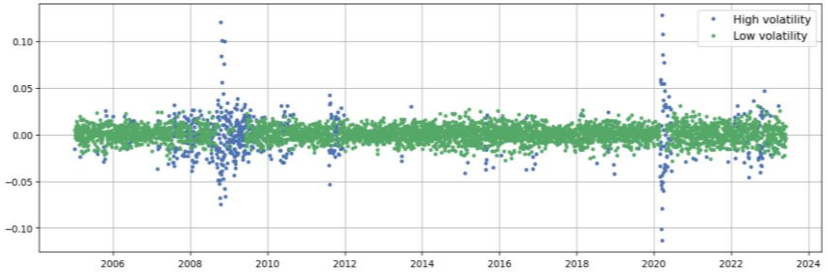


A2



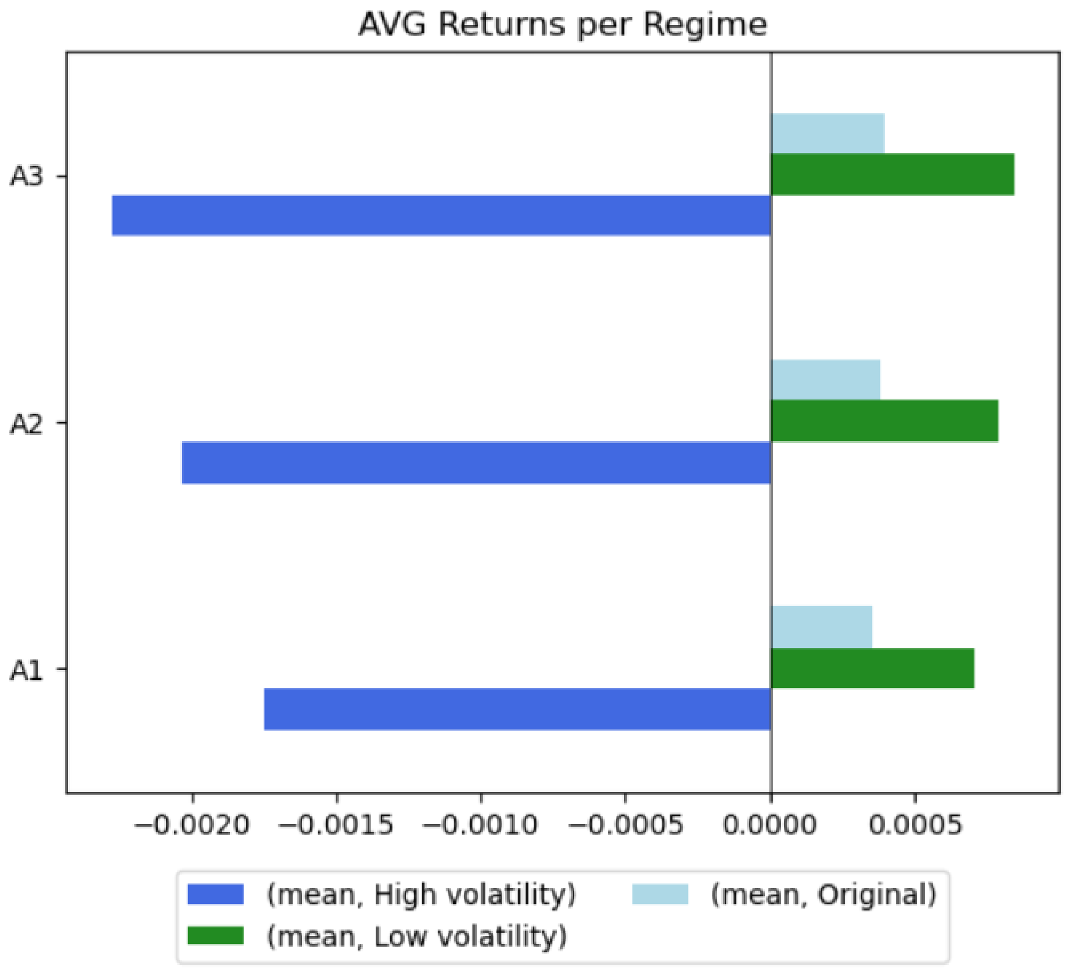
A3

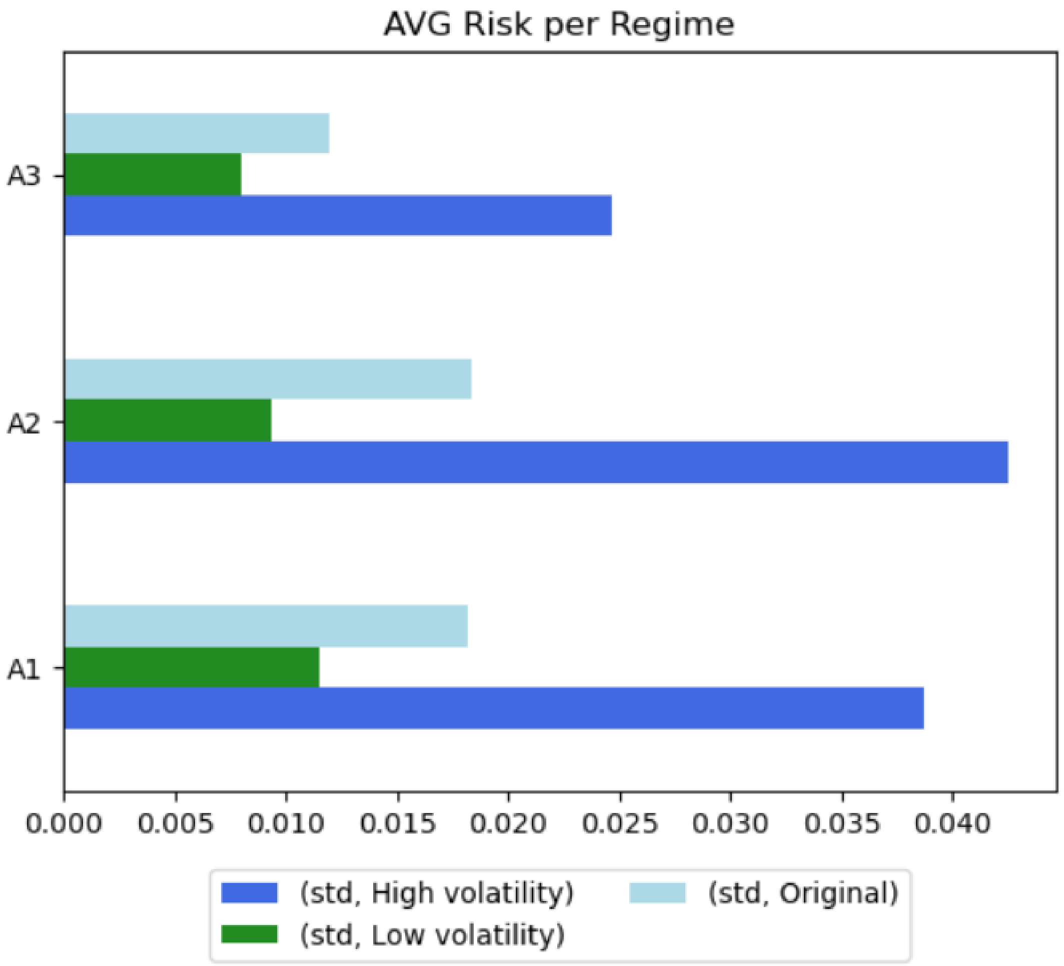




Asset returns based on Hidden States

A3 is outperforming A1 and A2 in both high- and low-volatility states

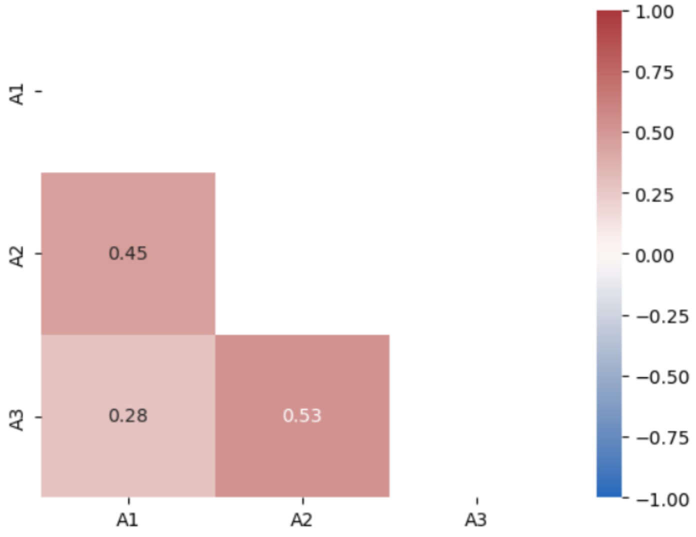




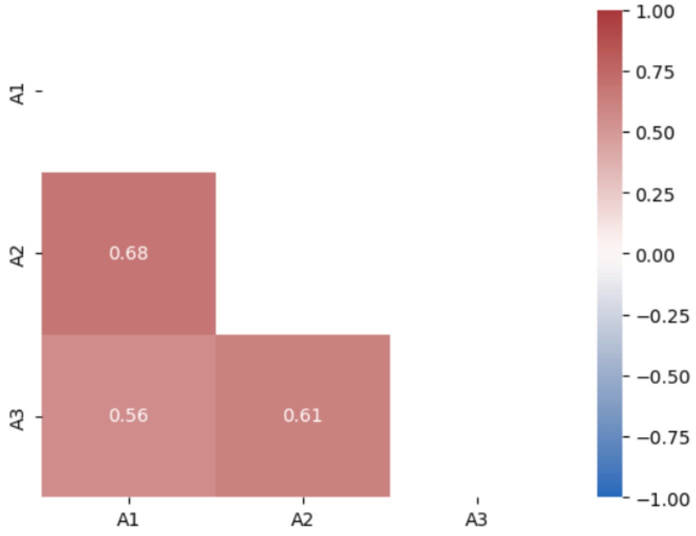
Correlation between assets

Assets' correlation changes according to the market states

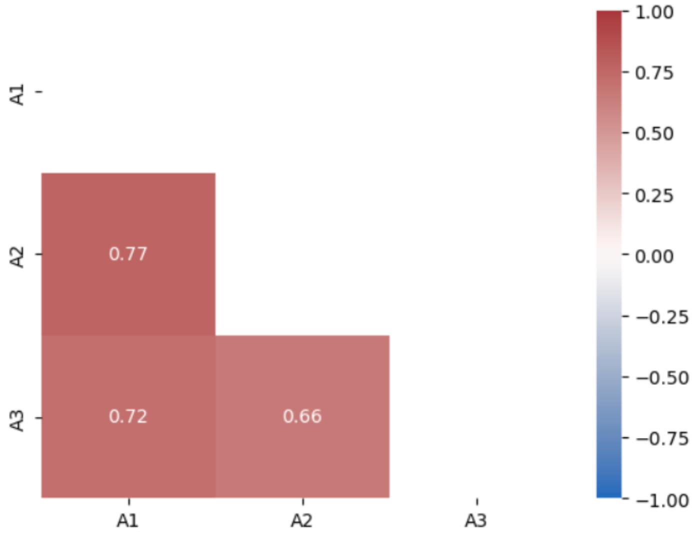
Low-Volatility State



Market State



High-Volatility State



Historical Returns

A3 has historically outperformed the other two assets

Covariance Matrix

Low-Volatility State

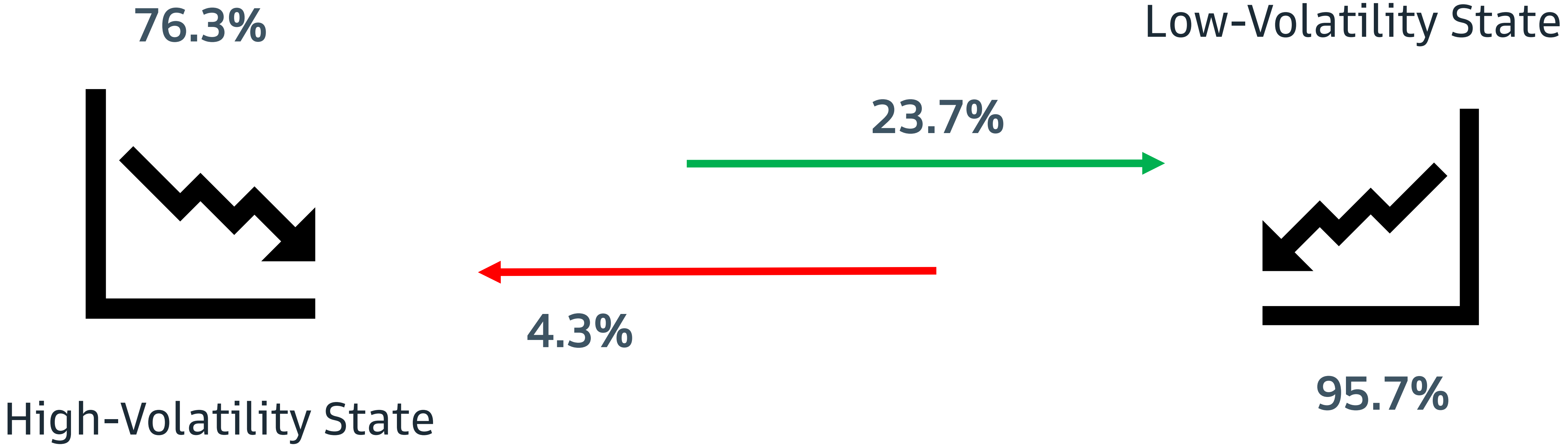
	A1	A2	A3
A1	0.001500	0.001269	0.000685
A2	0.001269	0.001811	0.000694
A3	0.000685	0.000694	0.000608

High-Volatility State

	A1	A2	A3
A1	0.000134	0.000049	0.000026
A2	0.000049	0.000088	0.000040
A3	0.000026	0.000040	0.000064

Hidden State Transition Probabilities

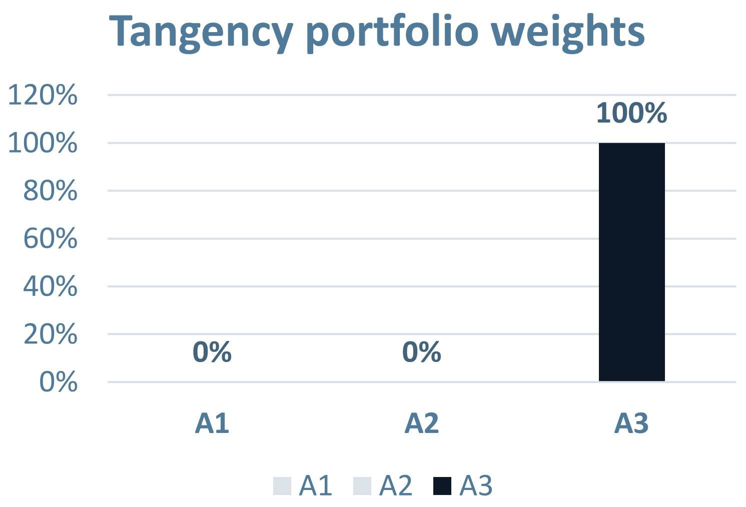
Using Hidden Markov Model (HMM), the hidden state transition probabilities refer to the probabilities of transitioning between different hidden states over time

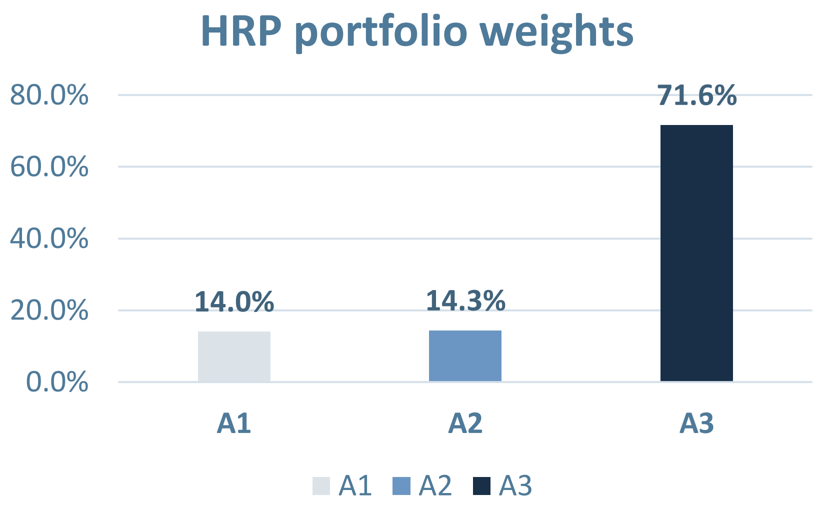


Portfolio optimization

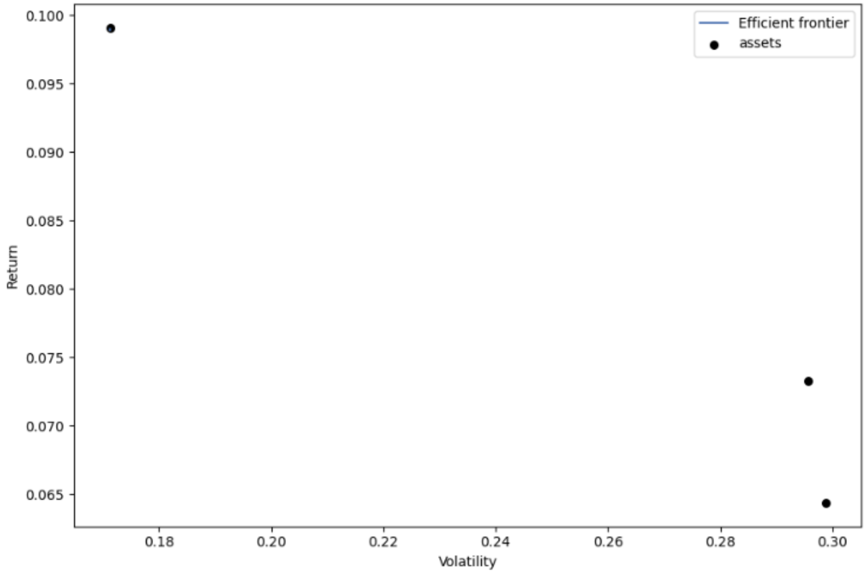
Overall

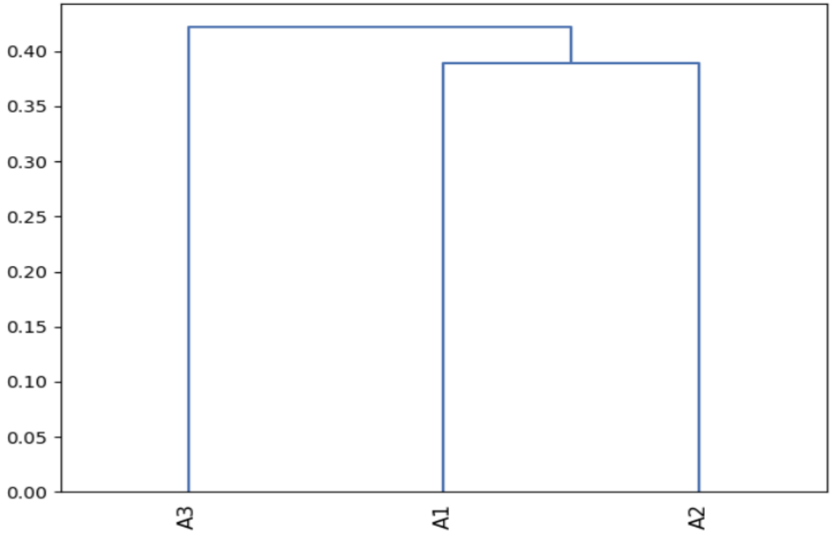
	Tangency (max. Sharpe ratio) portfolio	*Hierarchical Risk Parity (HRP) portfolio*
Expected annual return	9.9%	10.9%
Annual volatility	17.1%	18.2%
Sharpe Ratio	0.46	0.49

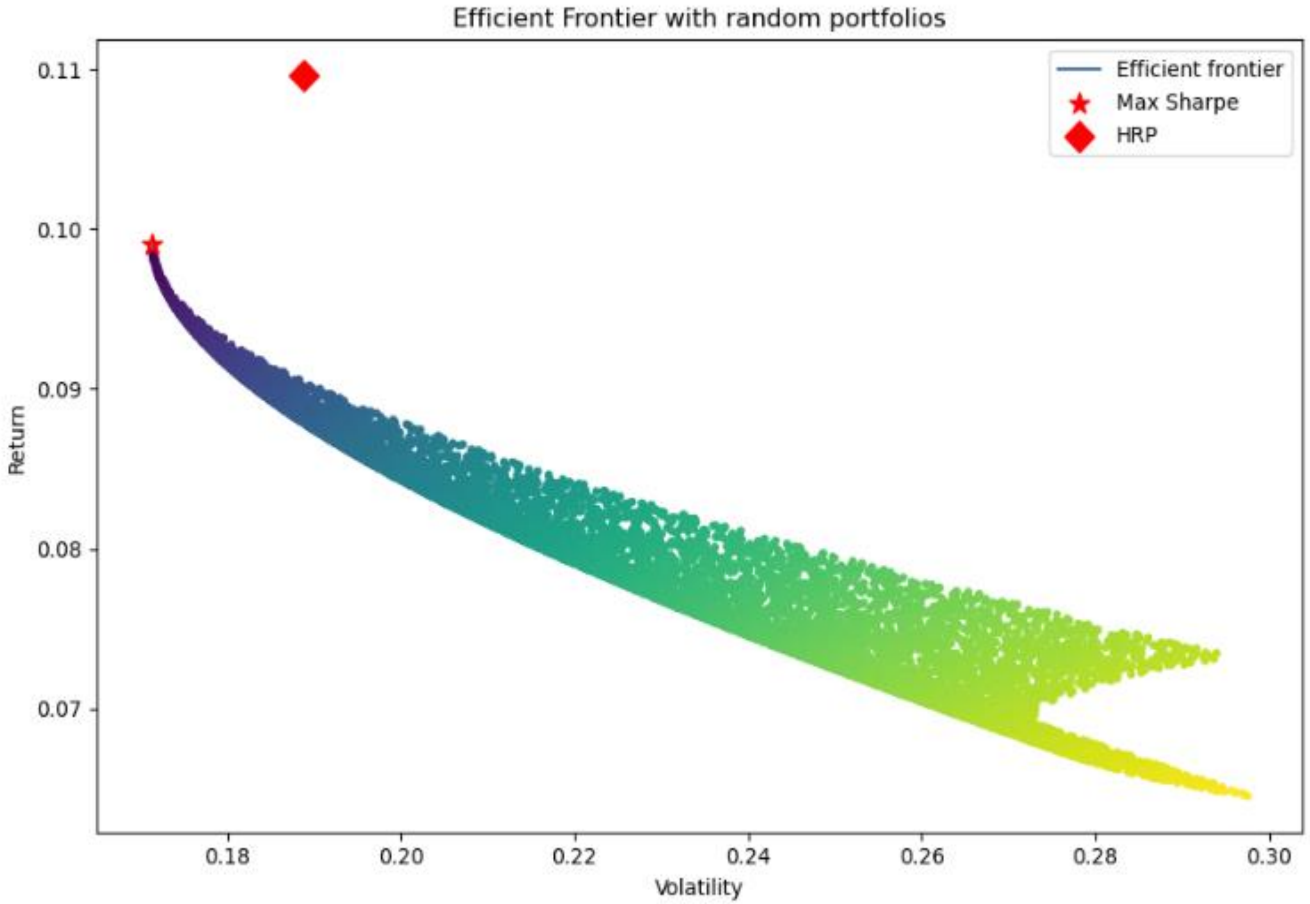




Portfolio optimization
Overall



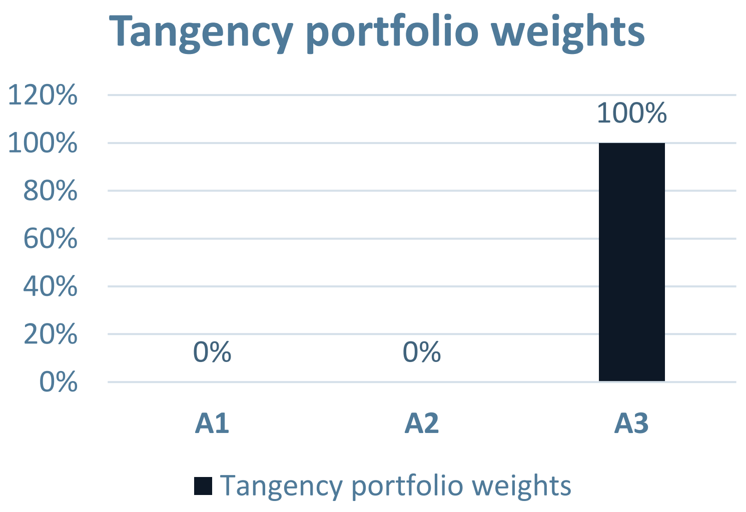


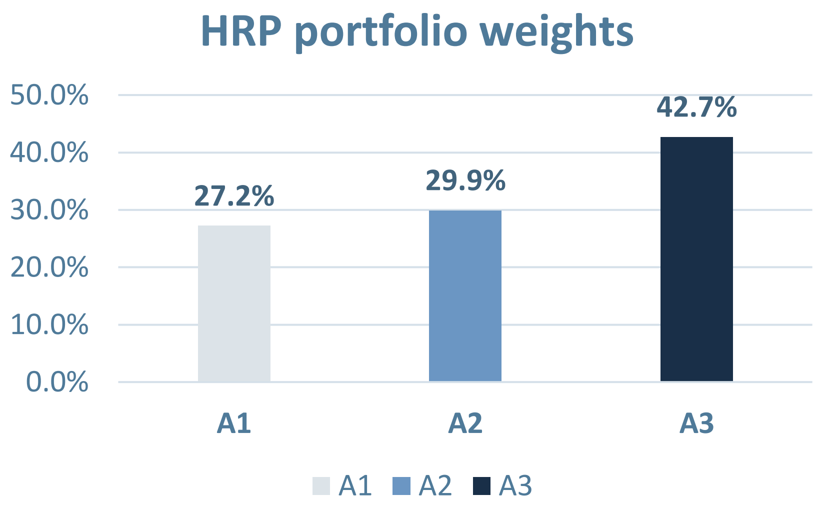


Portfolio optimization

Low Volatility

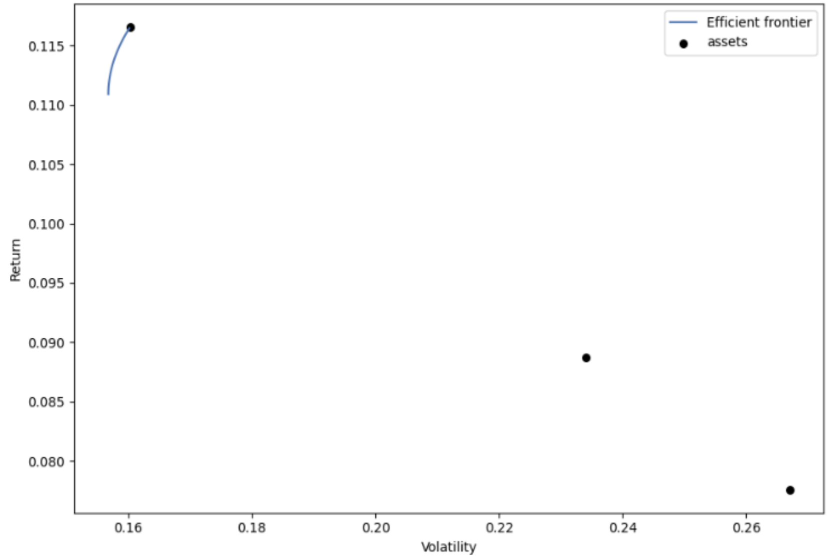
	Tangency (max. Sharpe ratio) portfolio	*Hierarchical Risk Parity (HRP) portfolio*
Expected annual return	11.7%	20.5%
Annual volatility	16.0%	11.5%
Sharpe Ratio	0.60	1.60

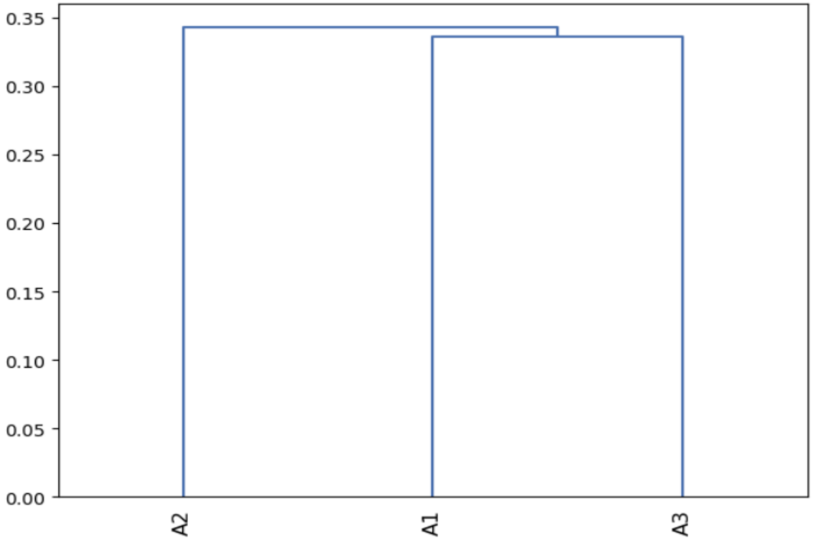


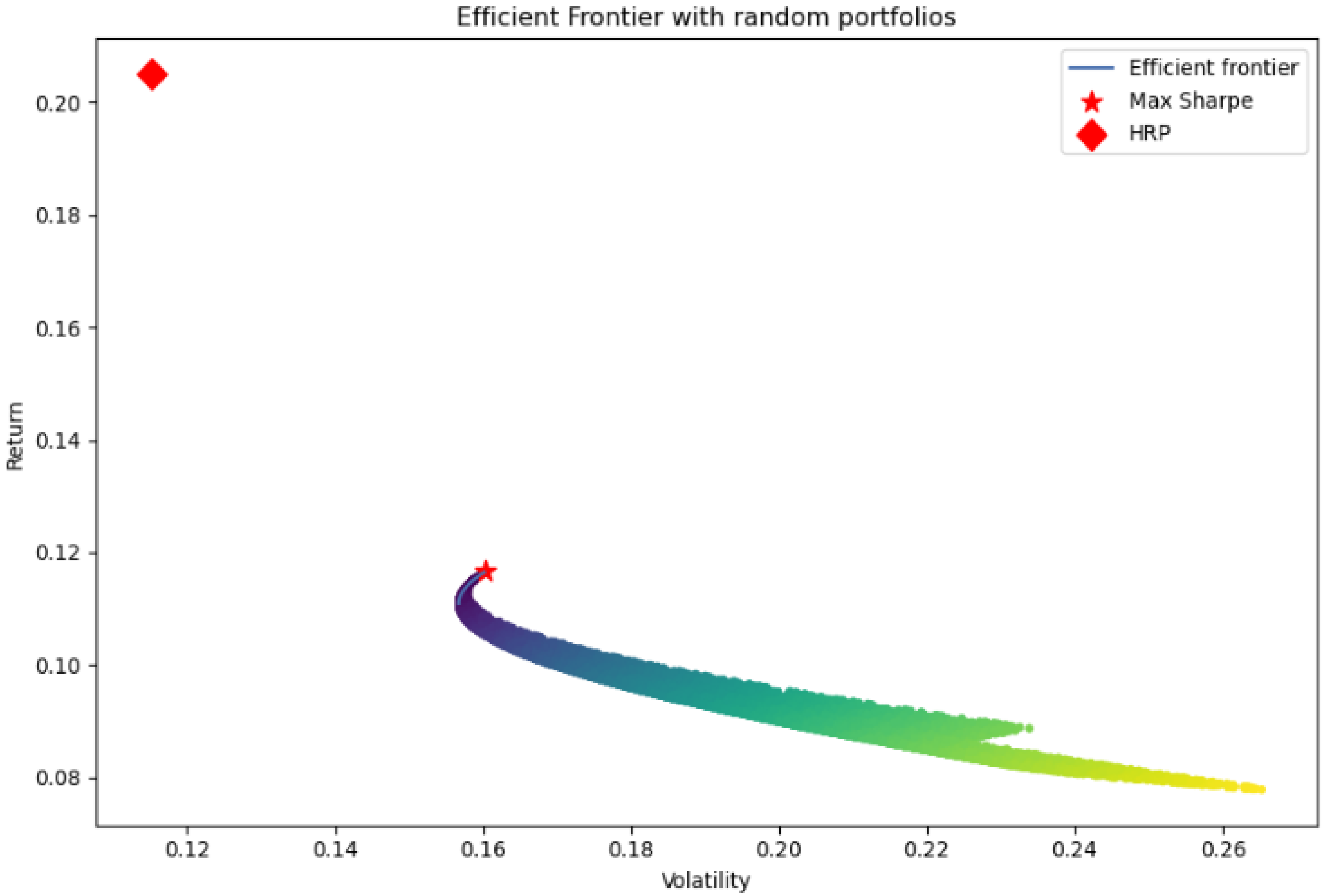


Portfolio optimization

Low Volatility





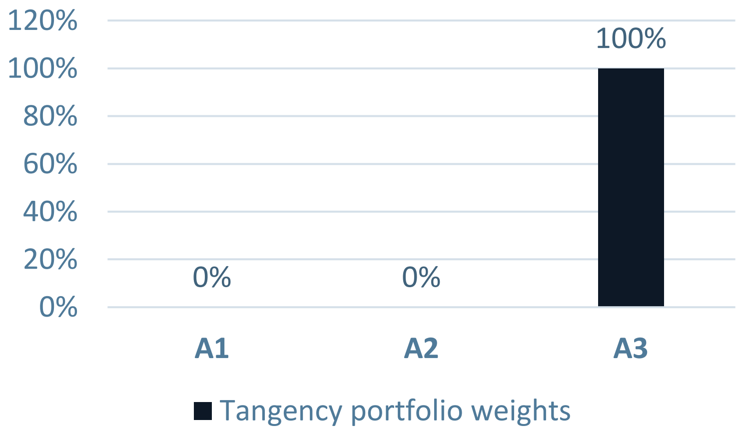


Portfolio optimization

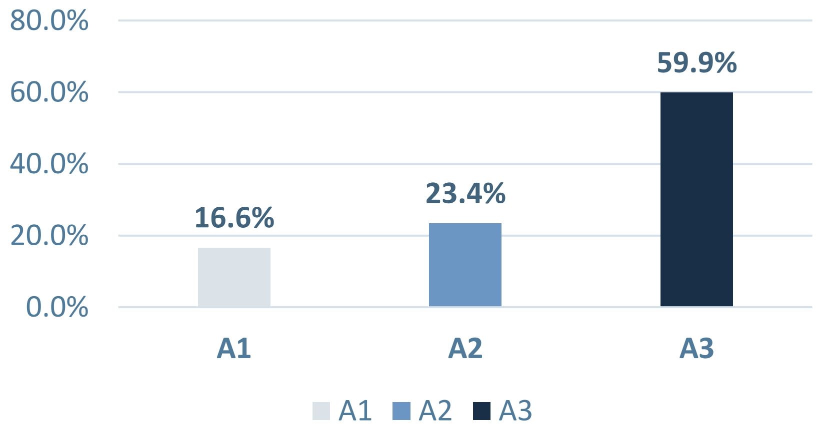
High Volatility

	Tangency (max. Sharpe ratio) portfolio	*Hierarchical Risk Parity (HRP) portfolio*
Expected annual return	100.9%	48.7%
Annual volatility	49.4%	42.6%
Sharpe Ratio	2.00	-1.19

Tangency portfolio weights

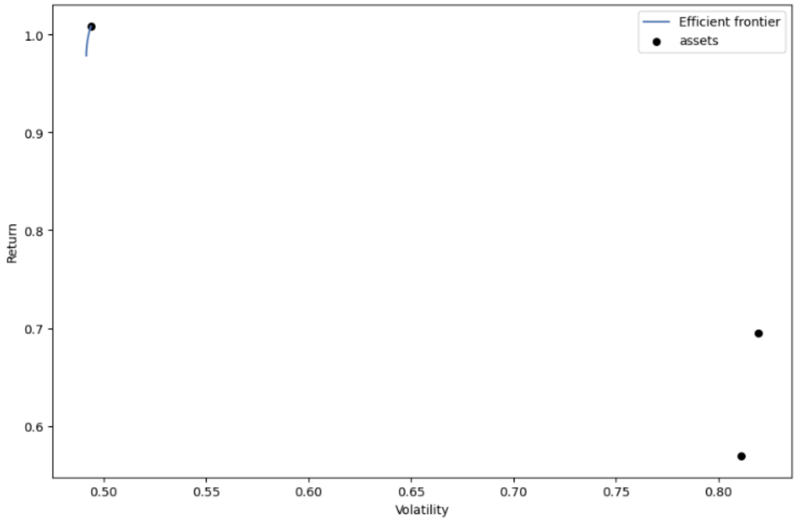


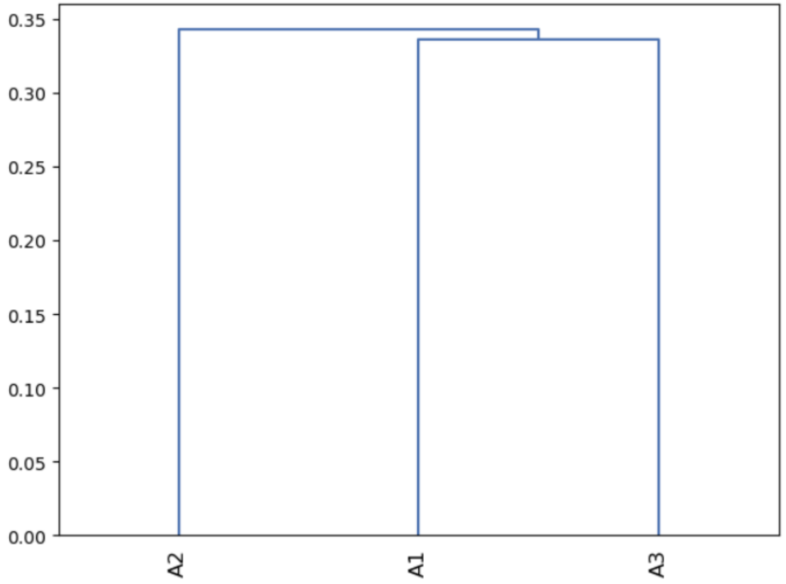
HRP portfolio weights

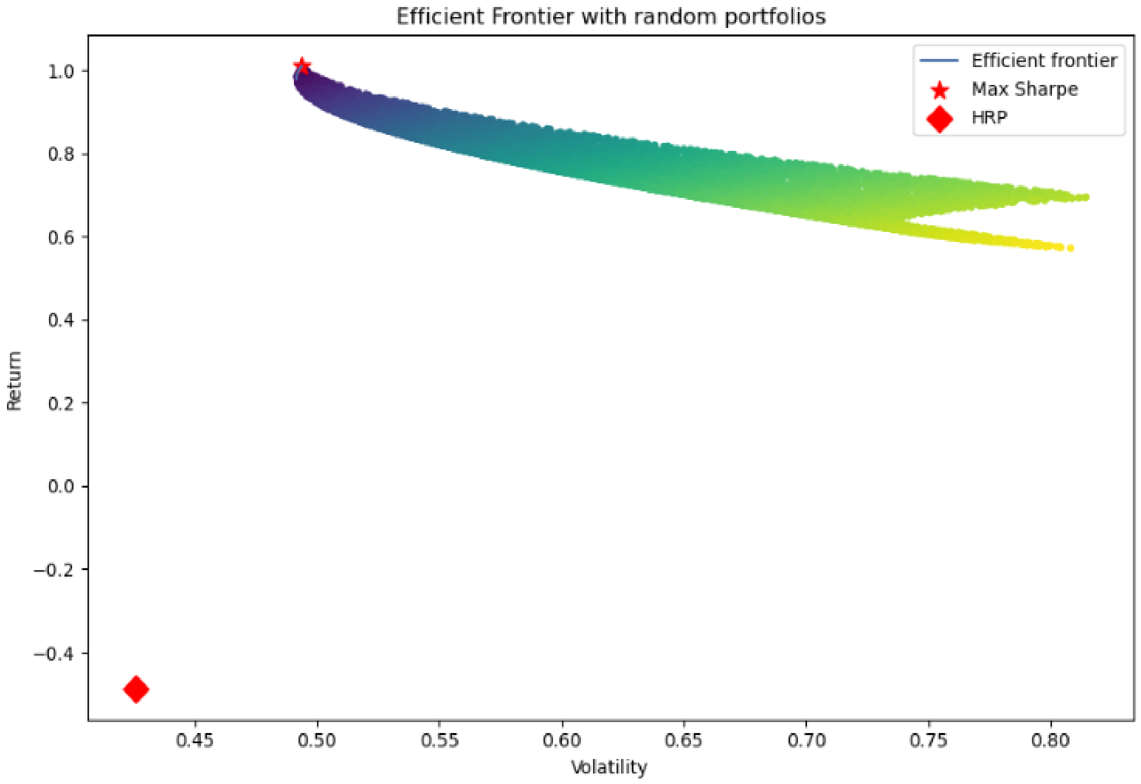


Portfolio optimization

High Volatility





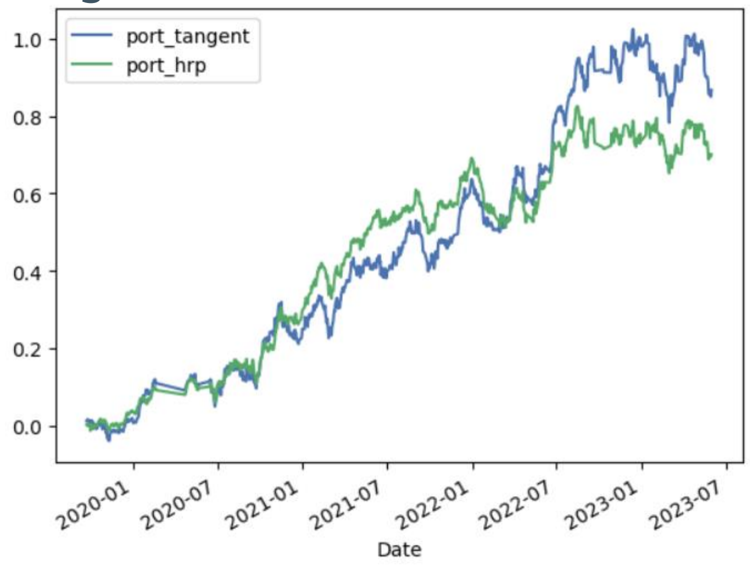


Findings and Recommendations

Maximum Sharpe portfolio appears to outperform in the majority of the identified states the HRP portfolio

Low-volatility State

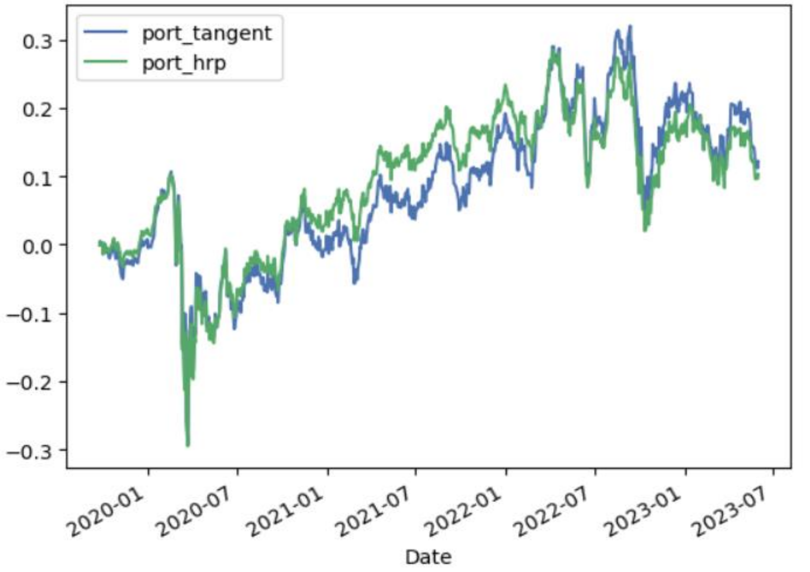
Tangent std: 0.317 HRP std: 0.257



Q3 and Q4 of 2022 represent the crucial point where the Maximum Sharpe portfolio outperforms the HRP

Original state

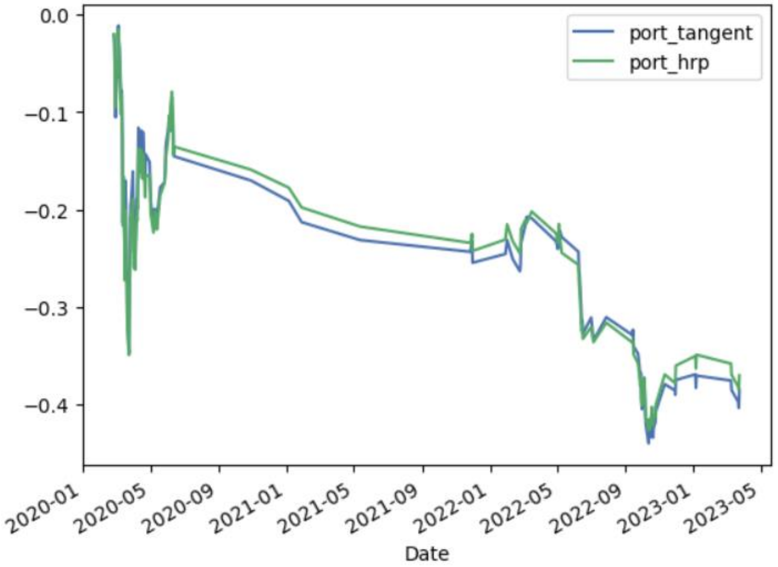
Tangent std: 0.193 HRP std: 0.173



The two strategies perform in a similar way throughout the analyzed time window

High-volatility State

Tangent std: 0.112 HRP std: 0.108



With few data points available in this state, the HRP slightly outperforms the Maximum Sharpe portfolio

HULT INTERNATIONAL BUSINESS SCHOOL

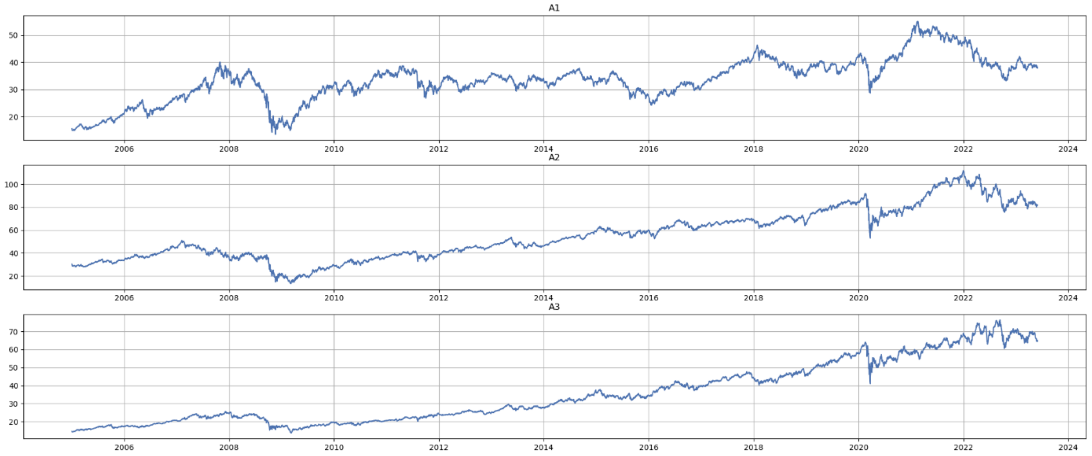
Thank you!

Appendix

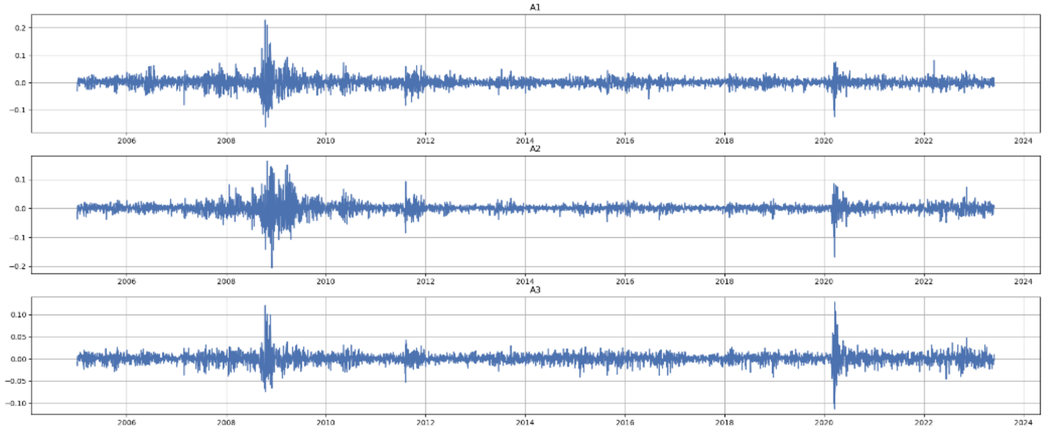
Main part of the code (Market Behaviour)

In [4]:
```
tickers = list(df.columns)
```

In [5]:
```
# asset prices
plt.figure(figsize = (25, 10))
plt.subplot(3,1,1)
plt.plot(df.index, df[tickers[0]])
plt.title(tickers[0])
plt.grid(True)
plt.subplot(3,1,2)
plt.plot(df.index, df[tickers[1]])
plt.title(tickers[1])
plt.grid(True)
plt.subplot(3,1,3)
plt.plot(df.index, df[tickers[2]])
plt.title(tickers[2])
plt.grid(True)
plt.show()
```



In [6]:
```
# daily returns
plt.figure(figsize = (25, 10))
plt.subplot(3,1,1)
plt.plot(df.index, df[tickers[0]].pct_change())
plt.title(tickers[0])
plt.grid(True)
plt.subplot(3,1,2)
plt.plot(df.index, df[tickers[1]].pct_change())
plt.title(tickers[1])
plt.grid(True)
plt.subplot(3,1,3)
plt.plot(df.index, df[tickers[2]].pct_change())
plt.title(tickers[2])
plt.grid(True)
plt.show()
```



Main part of the code (Hidden states of Markovitz Model)

In [14]:
```python
# Define status labels
states_dict = {0:'High volatility', 1:'Low volatility', 2:'Original'}
```

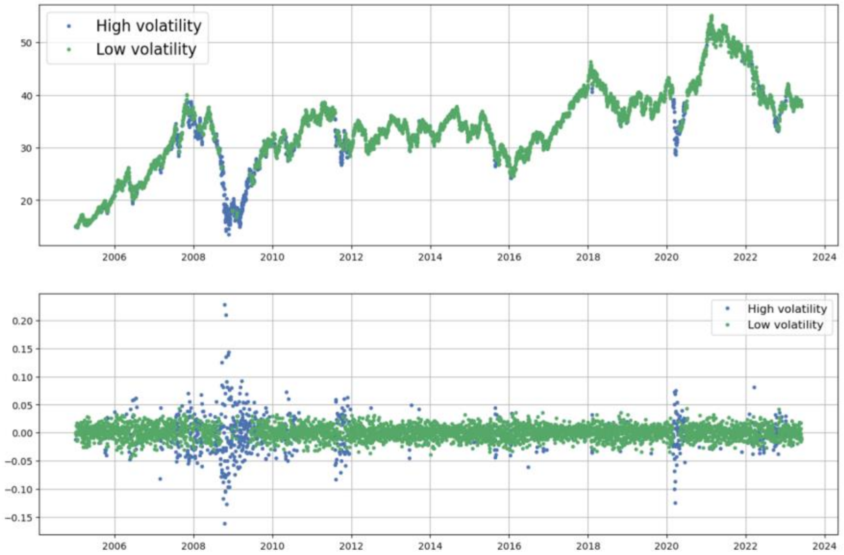
In [15]:
```python
pd.DataFrame(data=model.means_, index=[states_dict[0], states_dict[1]], columns=['A1', 'A2', 'A3'])
```

Out[15]:

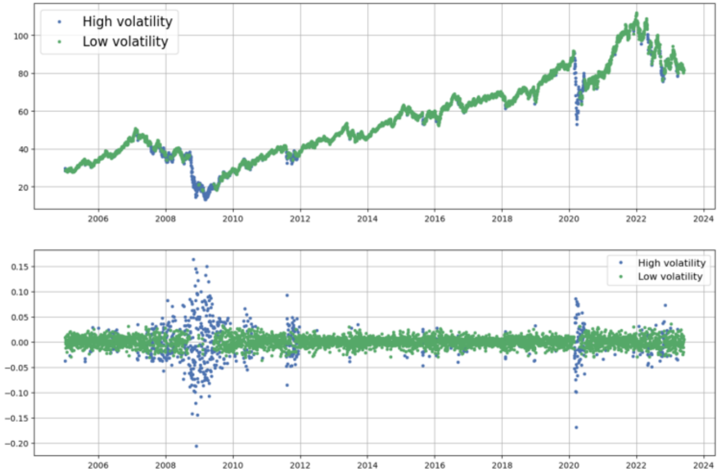
	A1	A2	A3
High volatility	-0.001679	-0.002035	-0.002186
Low volatility	0.000728	0.000826	0.000866

In [16]:
```python
# regime plotting function (will be called repeatedly for each asset)
def regime_plot(df, nasset):
    plt.figure(figsize=(15, 10))
    plt.subplot(2, 1, 1)

    for i in states:
        s = (Z == i)
        x = df.index[s]
        y = df[tickers[nasset]].iloc[s]
        plt.plot(x, y, '.', label=states_dict[i])  # Add label based on regime
    plt.legend(fontsize=16)
    plt.grid(True)

    plt.subplot(2, 1, 2)
    for i in states:
        s = (Z == i)
        x = df.index[s]
        y = df[tickers[nasset]].pct_change().iloc[s]
        plt.plot(x, y, '.', label=states_dict[i])  # Add label based on regime
    plt.legend(fontsize=12)
    plt.grid(True)

    return plt
```

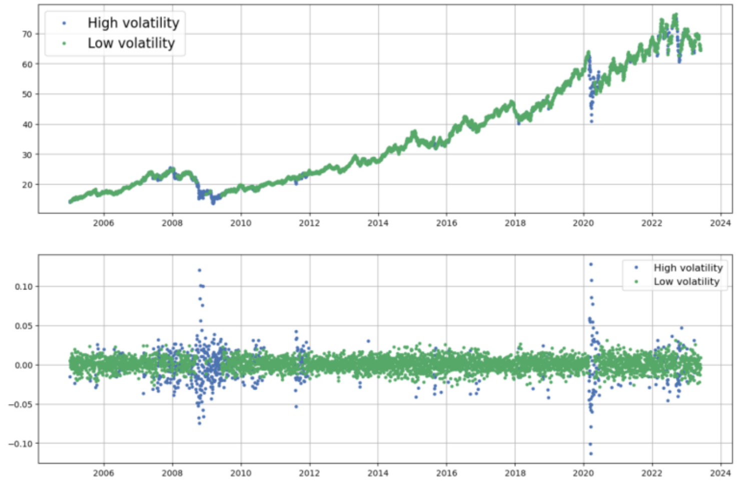
In [17]:
```python
# removing the first row of the data that has NA-return
regime_plot(df.iloc[1: , :], 0).show()
```



In [18]:
```python
regime_plot(df.iloc[1: , :], 1).show()
```



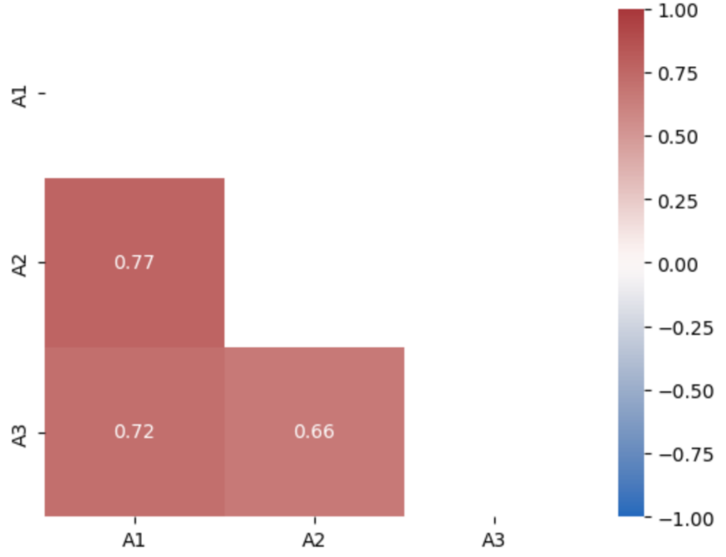
In [19]:
```python
regime_plot(df.iloc[1: , :], 2).show()
```



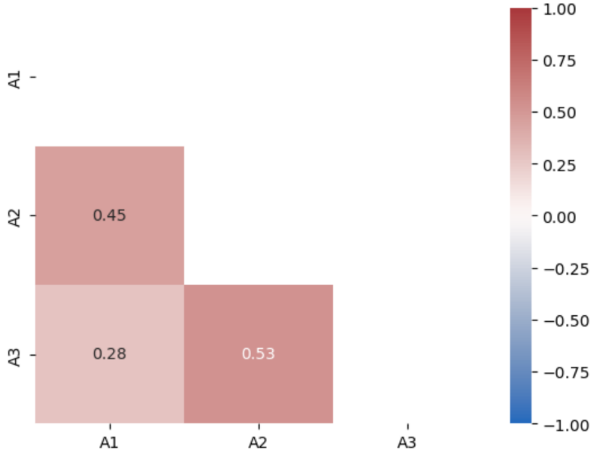
Main part of the code (Correlation between assets)

In [25]:
```python
low_vol_avg_returns = pd.DataFrame(low_vol_stats.loc["mean"])
low_vol_avg_returns["state"] = states_dict[1]

high_vol_avg_returns = pd.DataFrame(high_vol_stats.loc["mean"])
high_vol_avg_returns["state"] = states_dict[0]

original_avg_returns = pd.DataFrame(original_stats.loc["mean"])
original_avg_returns["state"] = states_dict[2]

avg_returns_df = pd.concat([low_vol_avg_returns, high_vol_avg_returns,original_avg_returns]).sort_index()
avg_returns_df.drop("state", inplace = True)

ax = avg_returns_df.pivot(columns='state').plot.barh()
plt.title('AVG Returns per Regime')

# Add a line at 0
ax.axvline(0, color='black', linewidth=0.5)
# Add a line at each maximum value
plt.show()
```

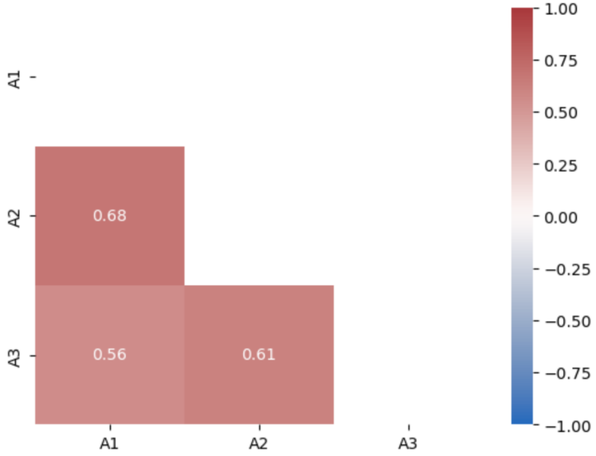
In [27]:
```python
corr_matrix = high_vol_returns_df[["A1","A2","A3"]].corr()

mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Plot the correlation matrix as a heatmap
sns.heatmap(corr_matrix, annot=True, vmin=-1, vmax=1, cmap="vlag", mask=mask)
plt.show()
```



In [28]:
```python
corr_matrix = low_vol_returns_df[["A1","A2","A3"]].corr()

mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Plot the correlation matrix as a heatmap
sns.heatmap(corr_matrix, annot=True, vmin=-1, vmax=1, cmap="vlag", mask=mask)
plt.show()
```



In [29]:
```python
corr_matrix = states_returns_df[["A1","A2","A3"]].corr()

mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Plot the correlation matrix as a heatmap
sns.heatmap(corr_matrix, annot=True, vmin=-1, vmax=1, cmap="vlag", mask=mask)
plt.show()
```



Main part of the code (OVERALL)

```
In [30]:
low_vol_df_s = low_vol_df.copy()
low_vol_df_s = low_vol_df_s[['A1','A2','A3']]

low_vol_returns_df_s = low_vol_returns_df.copy()
low_vol_returns_df_s = low_vol_returns_df_s[['A1','A2','A3']]

high_vol_df_s = high_vol_df.copy()
high_vol_df_s = high_vol_df_s[['A1','A2','A3']]

high_vol_returns_df_s = high_vol_returns_df.copy()
high_vol_returns_df_s = high_vol_returns_df_s[['A1','A2','A3']]

states_df_s = states_df.copy()
states_df_s = states_df_s[['A1','A2','A3']]

states_returns_df_s = states_returns_df.copy()
states_returns_df_s = states_returns_df_s[['A1','A2','A3']]
```

```
In [31]:
# Splitting low_vol_df into train and test sets
low_vol_train, low_vol_test = train_test_split(low_vol_df_s, test_size=0.2, shuffle=False)
low_vol_returns_train, low_vol_returns_test = train_test_split(low_vol_returns_df_s, test_size=0.2, shuffle=False)

# Splitting high_vol_df into train and test sets
high_vol_train, high_vol_test = train_test_split(high_vol_df_s, test_size=0.2, shuffle=False)
high_vol_returns_train, high_vol_returns_test = train_test_split(high_vol_returns_df_s, test_size=0.2, shuffle=False)


# Splitting states_df into train and test sets
states_train, states_test = train_test_split(states_df_s, test_size=0.2, shuffle=False)
states_returns_train, states_returns_test = train_test_split(states_returns_df_s, test_size=0.2, shuffle=False)
```

```
In [32]:
# average historical returns
mu = expected_returns.mean_historical_return(states_train)
mu.sort_values(ascending=False)

Out[32]:
A3    0.099094
A2    0.073264
A1    0.064398
dtype: float64
```

```
In [33]:
# historical covariance matrix
S = risk_models.sample_cov(states_train)
S
```

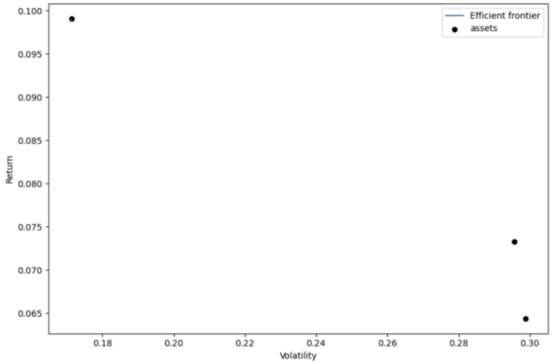
Out[33]:

	A1	A2	A3
A1	0.089234	0.060272	0.030934
A2	0.060272	0.087383	0.028964
A3	0.030934	0.028964	0.029349

```
In [34]:
# find efficient frontier
ef = EfficientFrontier(mu, S)
```

```
In [35]:
# save object copies for further calculations and plotting (auxiliary step)
ef_cp = ef.deepcopy()
ef_tangent = ef.deepcopy()
```

```
In [36]:
# plot efficient frontier: see PyPortfolioOpt docs
# https://pyportfolioopt.readthedocs.io/en/latest/Plotting.html
fig, ax = plt.subplots(figsize=(9,6))
plotting.plot_efficient_frontier(ef, ax=ax, show_assets=True)
plt.show()
```




```
In [37]:
# find the tangency (max. Sharpe ratio) portfolio
ef_tangent.max_sharpe()
ret_tangent, std_tangent, _ = ef_tangent.portfolio_performance(verbose=True)

Expected annual return: 9.9%
Annual volatility: 17.1%
Sharpe Ratio: 0.46
```

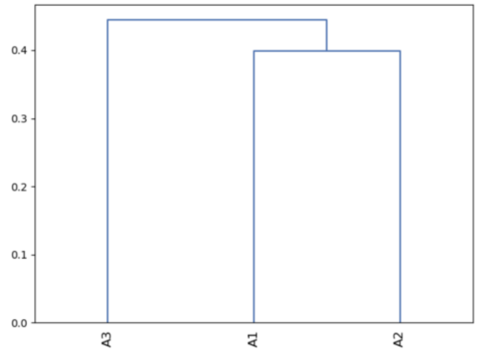
```
In [38]:
# tangency portfolio weights
tangent_weights = ef_tangent.clean_weights()
tangent_weights

Out[38]: OrderedDict([('A1', 0.0), ('A2', 0.0), ('A3', 1.0)])
```

```
In [39]:
# generate random portfolios for visualization
n_samples = 10000
w = np.random.dirichlet(np.ones(ef.n_assets), n_samples)
rets = w.dot(ef.expected_returns)
stds = np.sqrt(np.diag(w @ ef.cov_matrix @ w.T))
sharpes = rets / stds
sharpes

Out[39]: array([0.25608359, 0.47823743, 0.30080144, ..., 0.52275217, 0.26077736,
       0.38856064])
```

```
In [40]:
# Hierarchical Risk Parity (HRP) portfolio
hrp = HRPOpt(states_returns_train)
hrp_weights = hrp.optimize()
hrp_weights

Out[40]: OrderedDict([('A1', 0.14004440170405413),
             ('A2', 0.14309429590602324),
             ('A3', 0.7168613023899226)])
```

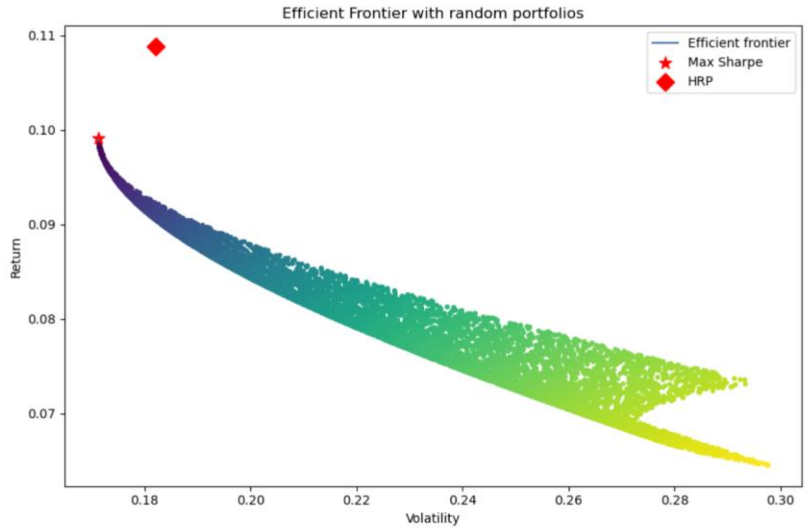
```
In [41]:
ret_hrp, std_hrp, _ = hrp.portfolio_performance(verbose=True)

Expected annual return: 10.9%
Annual volatility: 18.2%
Sharpe Ratio: 0.49
```

```
In [42]:
# Hierarchical Clustering dendrogram
plotting.plot_dendrogram(hrp, showfig = True)
```




```
Out[42]: <AxesSubplot:>
```

```
In [43]:
fig, ax = plt.subplots(figsize=(9,6))
plotting.plot_efficient_frontier(ef_cp, ax=ax, show_assets=False)

ax.scatter(stds, rets, marker=".", c=sharpes, cmap="viridis_r")

ax.scatter(std_tangent, ret_tangent, marker="*", s=100, c="r", label="Max Sharpe")

ax.scatter(std_hrp, ret_hrp, marker="D", s=100, c="r", label="HRP")

ax.set_title("Efficient Frontier with random portfolios")
ax.legend()
plt.tight_layout()
plt.show()
```



Main part of the code (Findings and Recommendations)

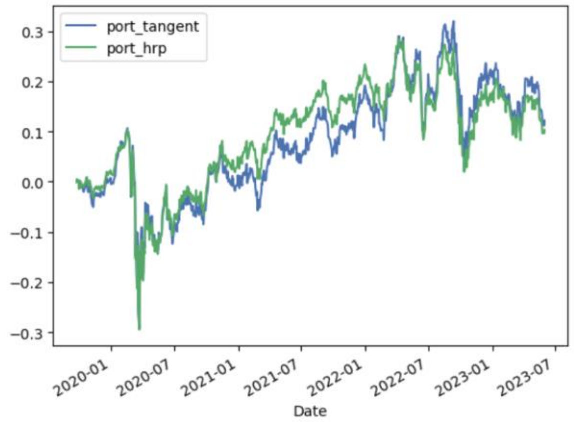
In [44]:
```
# construct portfolio returns: testing time period
states_returns_test['port_tangent'] = 0
for ticker, weight in tangent_weights.items():
    states_returns_test['port_tangent'] += states_returns_test[ticker]*weight
```

In [45]:
```
states_returns_test['port_hrp'] = 0
for ticker, weight in hrp_weights.items():
    states_returns_test['port_hrp'] += states_returns_test[ticker]*weight
```

In [46]:
```
# cumulative equity curve (recall from the financial data practice earlier)
port_equity_tangent = (1 + states_returns_test['port_tangent']).cumprod() - 1
port_equity_hrp = (1 + states_returns_test['port_hrp']).cumprod() - 1
port_equity = port_equity_tangent.to_frame().join(port_equity_hrp)
port_equity
```

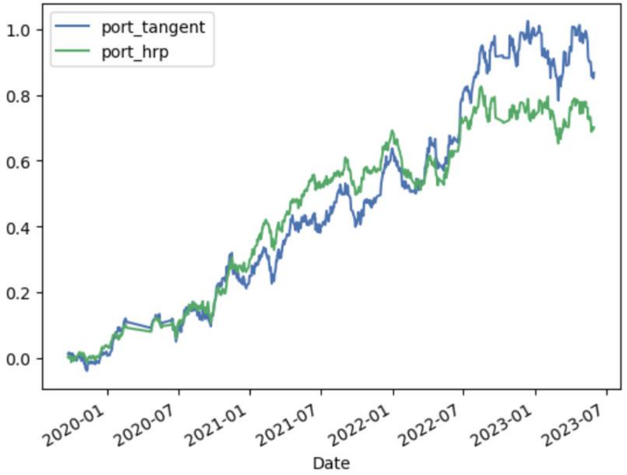
In [48]:
```
# out-of-sample performance
port_equity.plot()
```

Out[48]: <AxesSubplot:xlabel='Date'>



In [66]:
```
# out-of-sample performance
port_equity.plot()
```

Out[66]: <AxesSubplot:xlabel='Date'>



In [80]:
```
# construct portfolio returns: testing time period
high_vol_returns_test['port_tangent'] = 0
for ticker, weight in tangent_weights.items():
    high_vol_returns_test['port_tangent'] += high_vol_returns_test[ticker]*weight
```

In [81]:
```
high_vol_returns_test['port_hrp'] = 0
for ticker, weight in hrp_weights.items():
    high_vol_returns_test['port_hrp'] += high_vol_returns_test[ticker]*weight
```

In [82]:
```
# cumulative equity curve (recall from the financial data practice earlier)
port_equity_tangent = (1 + high_vol_returns_test['port_tangent']).cumprod() - 1
port_equity_hrp = (1 + high_vol_returns_test['port_hrp']).cumprod() - 1
port_equity = port_equity_tangent.to_frame().join(port_equity_hrp)
port_equity
```

In [84]:
```
# out-of-sample performance
port_equity.plot()
```

Out[84]: <AxesSubplot:xlabel='Date'>