

# A.2 AI For Investment Management Team 4

## Data Exploration

```
In [2]: # Package Installation
# https://anaconda.org/conda-forge/hmmLearn
# pip% install ____
```

```
In [1]: # Importing the required Dataset and Libraries
from pypfopt import EfficientFrontier
from pypfopt import risk_models
from pypfopt import expected_returns
from pypfopt import HRPOpt
from pypfopt import plotting
from sklearn.model_selection import train_test_split
from hmmlearn import hmm
import copy
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [3]: # read asset prices dataset from Github into a pandas dataframe
# (MODIFY file name for your group!)
fcsv = "https://raw.githubusercontent.com/multidis/hult-ai-investment-management/main/
df = pd.read_csv(fcsv, index_col=[0], parse_dates=['Date'])
df
```

Out[3]:

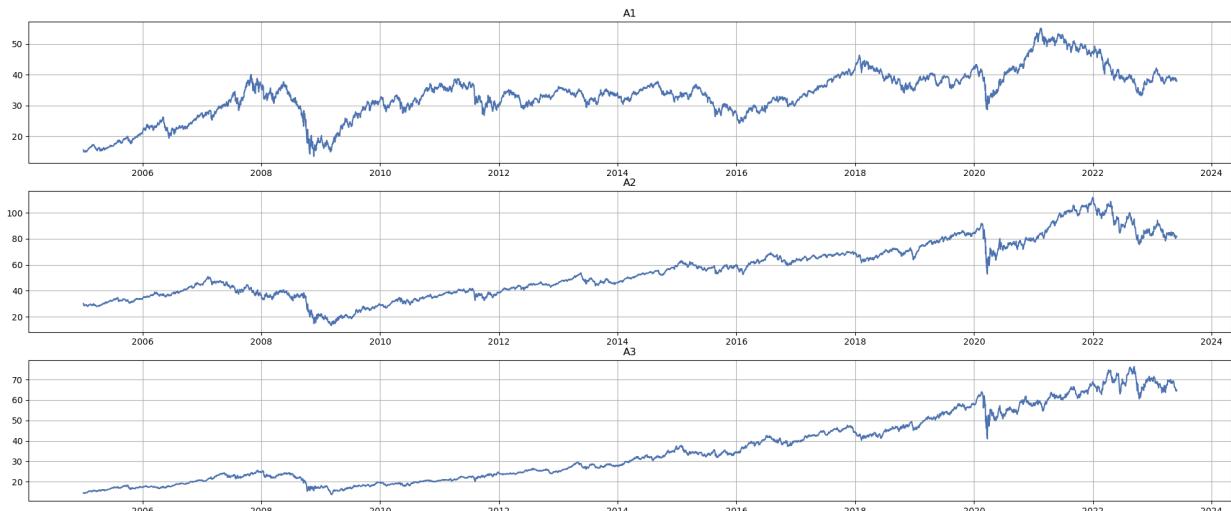
	A1	A2	A3
Date			
2005-01-03	15.559469	30.156376	14.526597
2005-01-04	15.080416	29.742468	14.426337
2005-01-05	14.895803	28.629606	14.199438
2005-01-06	14.885677	28.874975	14.273320
2005-01-07	14.914498	28.872492	14.273320
...	...	...	...
2023-05-24	38.101505	80.131714	65.470001
2023-05-25	38.022125	80.141663	64.589996
2023-05-26	38.696838	81.067291	64.589996
2023-05-30	38.250336	81.415657	64.339996
2023-05-31	37.893135	81.843636	64.930000

4634 rows × 3 columns

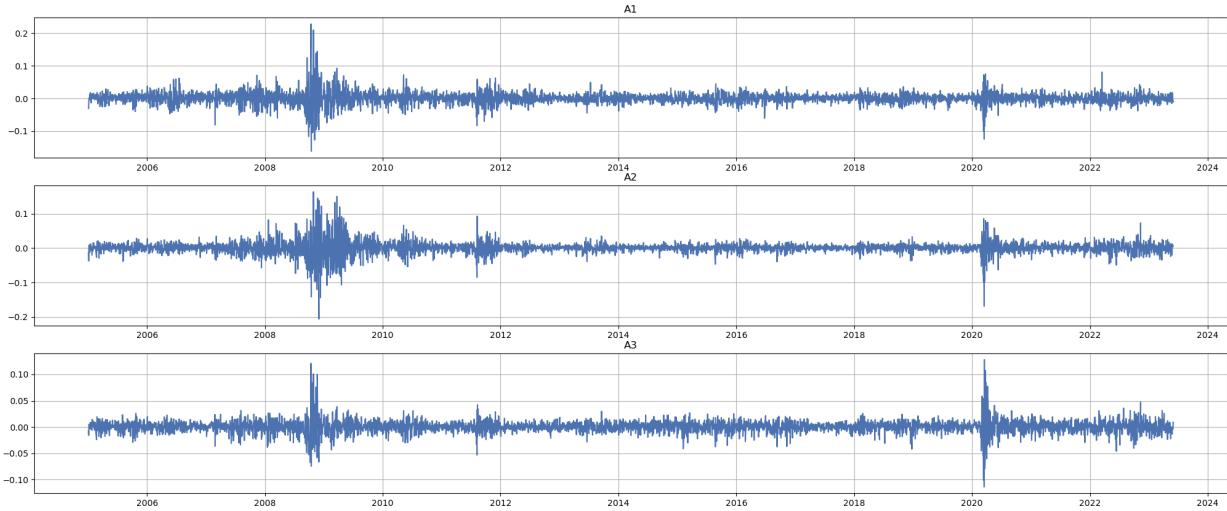
# Analyzing the market

```
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()
```



```
In [7]: def retcolumn(ticker):
    return df[ticker].pct_change().dropna().values
```

```
In [8]: # extract numerical values only from pandas dataframe (numpy array)
X = np.column_stack((retcolumn(tickers[0]), retcolumn(tickers[1]), retcolumn(tickers[2]),
X
```

```
Out[8]: array([[-0.03078855, -0.01372539, -0.00690181],
 [-0.01224186, -0.03741659, -0.01572812],
 [-0.0006798 ,  0.00857046,  0.00520317],
 ...,
 [ 0.01774528,  0.01154991,  0.        ],
 [-0.01153848,  0.00429724, -0.00387057],
 [-0.0093385 ,  0.00525671,  0.0091701 ]])
```

## Modelling

```
In [9]: # Learning the hidden states (HMM model fit)
model = hmm.GaussianHMM(n_components = 2, covariance_type = "diag", n_iter = 50, random_state=15)
model
```

```
Out[9]: GaussianHMM(n_components=2, n_iter=50, random_state=15)
```

```
In [10]: model.fit(X)
```

```
Out[10]: GaussianHMM(n_components=2, n_iter=50, random_state=15)
```

```
In [11]: # hidden states corresponding to observed X
Z = model.predict(X)
Z
```

```
Out[11]: array([0, 0, 1, ..., 1, 1, 1], dtype=int64)
```

```
In [12]: # distinct states
states = pd.unique(Z)
states
```

```
Out[12]: array([0, 1], dtype=int64)
```

```
In [13]: # hidden state transition probabilities
model.transmat_
```

```
Out[13]: array([[0.76311961, 0.23688039],
 [0.04299079, 0.95700921]])
```

## Defining status labels

High Volatility State: Market condition with large and frequent price fluctuations, indicating increased risk and uncertainty.

Low Volatility State: Market condition with small and infrequent price movements, indicating stability and reduced risk.

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

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

	A1	A2	A3
<b>High volatility</b>	-0.001679	-0.002035	-0.002186
<b>Low volatility</b>	0.000728	0.000826	0.000866

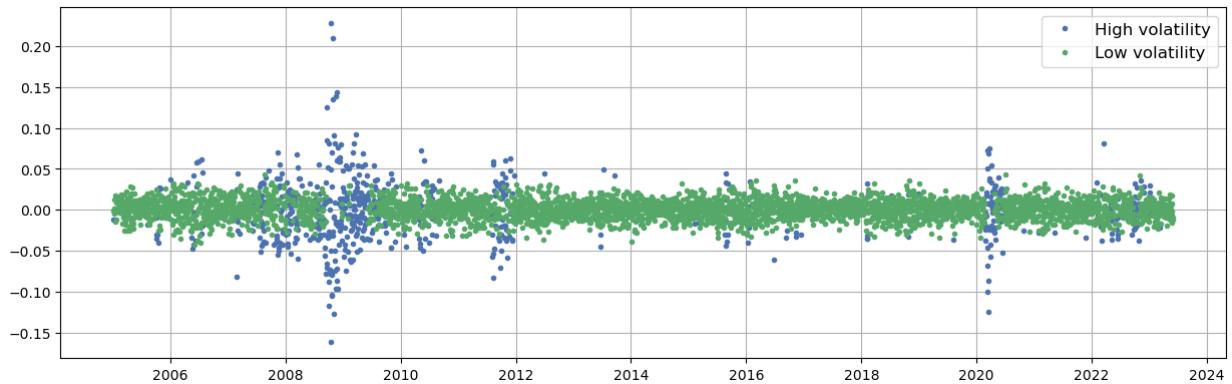
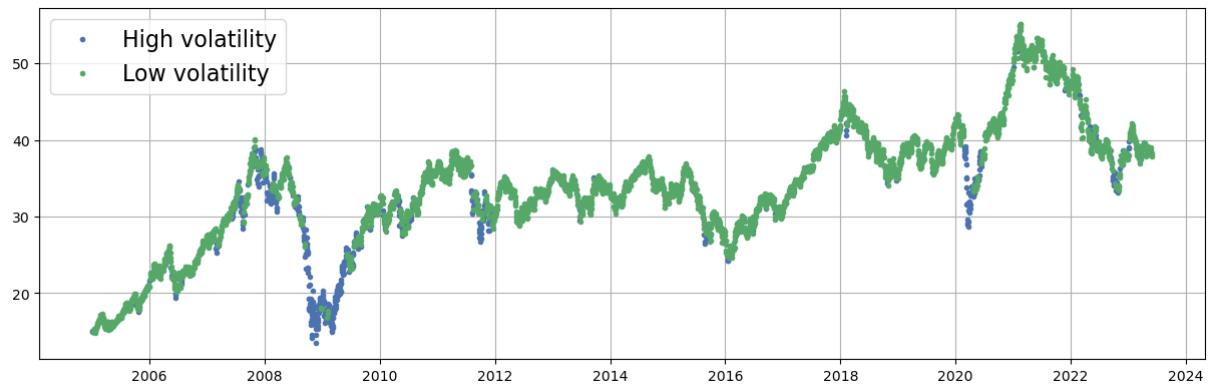
```
In [16]: # regime plotting function (will be called repeatedly for each asset)
def regime_plot(df, nasset):
    plt.figure(figsize=(15, 10))
    plt.subplot(2, 1, 1)

    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)

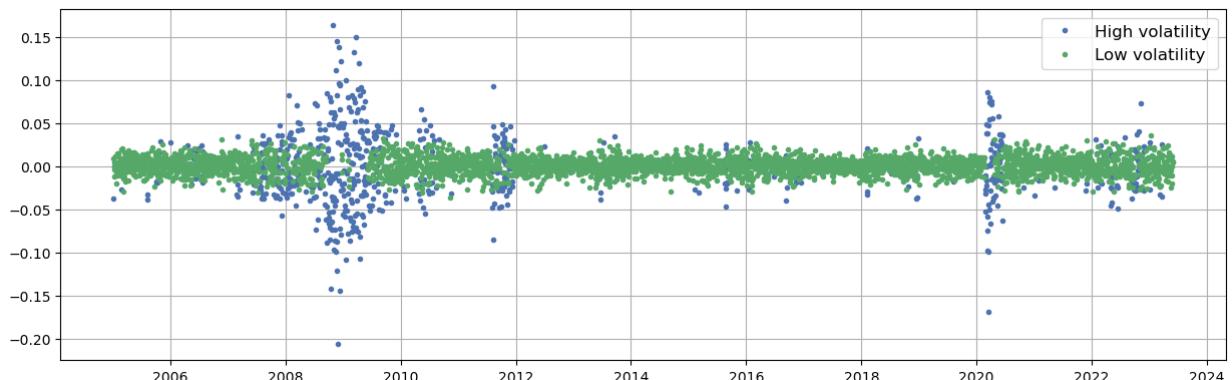
    plt.subplot(2, 1, 2)
    for i in states:
        s = (Z == i)
        x = df.index[s]
        y = df[tickers[nasset]].pct_change().iloc[s]
        plt.plot(x, y, '.', label=states_dict[i]) # Add label based on regime
    plt.legend(fontsize=12)
    plt.grid(True)

    return plt
```

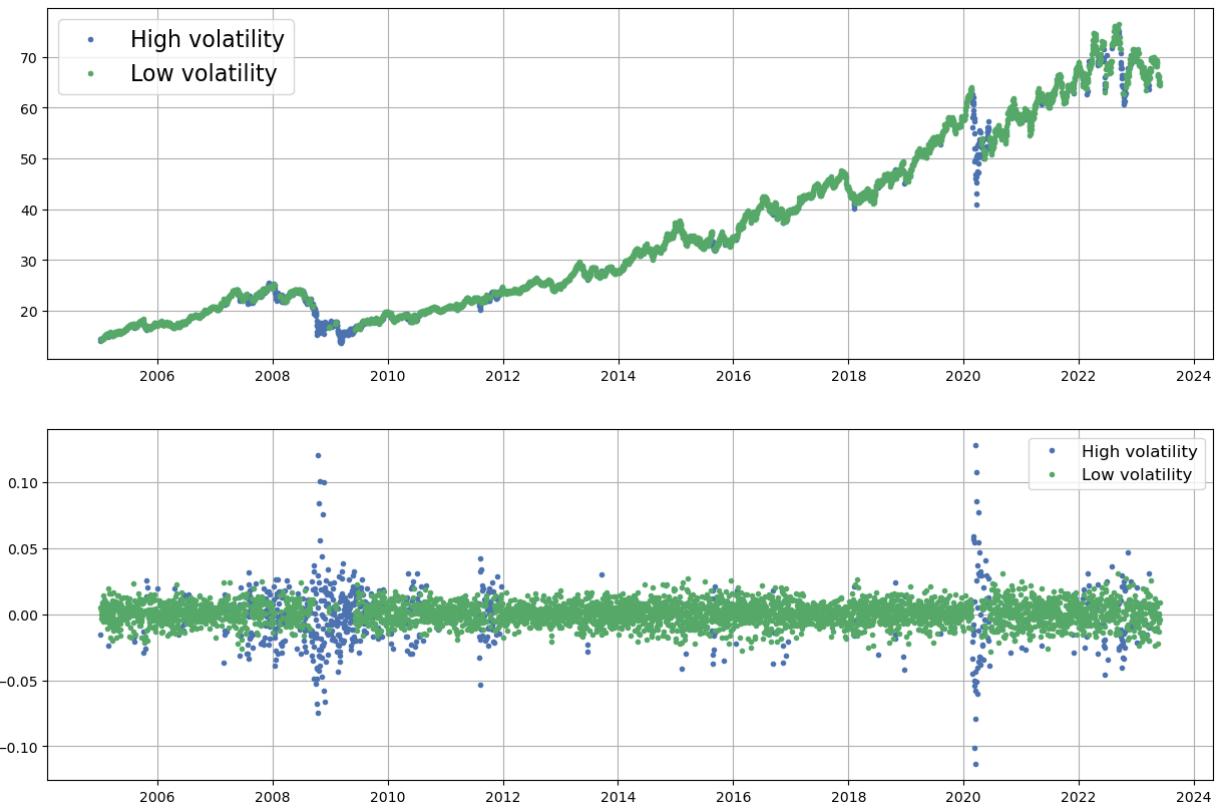
```
In [17]: # removing the first row of the data that has NA-return
regime_plot(df.iloc[1: , :], 0).show()
```



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



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



```
In [20]: # Calculating the returns of the market
states_df = df.copy()
states_returns_df = df.copy()
states_returns_df = states_df.pct_change().dropna()

# Adding a column that specifies the State for each date
states_df = states_df.iloc[1:]
states_df['state'] = Z
states_returns_df['state'] = Z

states_df, states_returns_df
```

```
Out[20]: (
```

	A1	A2	A3	state
Date				
2005-01-04	15.080416	29.742468	14.426337	0
2005-01-05	14.895803	28.629606	14.199438	0
2005-01-06	14.885677	28.874975	14.273320	1
2005-01-07	14.914498	28.872492	14.273320	1
2005-01-10	14.933196	28.855146	14.347193	1
...	...	...	...	...
2023-05-24	38.101505	80.131714	65.470001	1
2023-05-25	38.022125	80.141663	64.589996	1
2023-05-26	38.696838	81.067291	64.589996	1
2023-05-30	38.250336	81.415657	64.339996	1
2023-05-31	37.893135	81.843636	64.930000	1

[4633 rows x 4 columns],

	A1	A2	A3	state
Date				
2005-01-04	-0.030789	-0.013725	-0.006902	0
2005-01-05	-0.012242	-0.037417	-0.015728	0
2005-01-06	-0.000680	0.008570	0.005203	1
2005-01-07	0.001936	-0.000086	0.000000	1
2005-01-10	0.001254	-0.000601	0.005176	1
...	...	...	...	...
2023-05-24	-0.007495	-0.020441	-0.006525	1
2023-05-25	-0.002083	0.000124	-0.013441	1
2023-05-26	0.017745	0.011550	0.000000	1
2023-05-30	-0.011538	0.004297	-0.003871	1
2023-05-31	-0.009338	0.005257	0.009170	1

[4633 rows x 4 columns])

```
In [21]: # EXAMPLE: average returns in the first market state (Z == 0)
df.pct_change().dropna().iloc[(Z == 0)].cov()
```

```
Out[21]:
```

	A1	A2	A3
<b>A1</b>	0.001500	0.001269	0.000685
<b>A2</b>	0.001269	0.001811	0.000694
<b>A3</b>	0.000685	0.000694	0.000608

```
In [22]: # EXAMPLE: covariance matrix for the second market state (Z == 1)
df.pct_change().dropna().iloc[(Z == 1)].cov()
```

```
Out[22]:
```

	A1	A2	A3
<b>A1</b>	0.000134	0.000049	0.000026
<b>A2</b>	0.000049	0.000088	0.000040
<b>A3</b>	0.000026	0.000040	0.000064

## Regime Analysis

```
In [23]: # Separating the original dataframe into high volatility and low volatility DataFrame
low_vol_df = states_df.loc[states_df['state'] == 1]
```

```
high_vol_df = states_df.loc[states_df['state'] == 0]

# Separate df into low volatility and high volatility
low_vol_returns_df = states_returns_df.loc[states_returns_df['state'] == 1]
high_vol_returns_df = states_returns_df.loc[states_returns_df['state'] == 0]

# Print Stats
low_vol_stats = low_vol_returns_df.describe()
high_vol_stats = high_vol_returns_df.describe()
original_stats = states_returns_df.describe()

low_vol_stats, high_vol_stats, original_stats
```

Out[23]:

	A1	A2	A3	state
count	3968.000000	3968.000000	3968.000000	3968.0
mean	0.000709	0.000789	0.000842	1.0
std	0.011559	0.009377	0.007990	0.0
min	-0.042443	-0.036101	-0.028064	1.0
25%	-0.006707	-0.004860	-0.004189	1.0
50%	0.000962	0.000995	0.001128	1.0
75%	0.008043	0.006583	0.005984	1.0
max	0.043037	0.036343	0.031288	1.0,
	A1	A2	A3	state
count	665.000000	665.000000	665.000000	665.0
mean	-0.001748	-0.002032	-0.002276	0.0
std	0.038725	0.042555	0.024658	0.0
min	-0.161662	-0.206111	-0.113577	0.0
25%	-0.026299	-0.027551	-0.017362	0.0
50%	-0.003048	-0.007849	-0.001825	0.0
75%	0.020044	0.025109	0.012732	0.0
max	0.227699	0.163253	0.127934	0.0,
	A1	A2	A3	state
count	4633.000000	4633.000000	4633.000000	4633.000000
mean	0.000357	0.000384	0.000395	0.856464
std	0.018170	0.018327	0.011959	0.350656
min	-0.161662	-0.206111	-0.113577	0.000000
25%	-0.008052	-0.006152	-0.005169	1.000000
50%	0.000901	0.000811	0.000966	1.000000
75%	0.008983	0.007457	0.006457	1.000000
max	0.227699	0.163253	0.127934	1.000000)

In [24]:

```
# Joining std and mean in the same graph to show market behavior
market_behavior = pd.DataFrame(original_stats.loc[['mean', 'std']])
market_behavior
```

Out[24]:

	A1	A2	A3	state
<b>mean</b>	0.000357	0.000384	0.000395	0.856464
<b>std</b>	0.018170	0.018327	0.011959	0.350656

In [25]:

```
# Separate the mean into a different DataFrame for each state of the market
low_vol_avg_returns = pd.DataFrame(low_vol_stats.loc["mean"])
low_vol_avg_returns["state"] = states_dict[1]

# Separate the mean into a different DataFrame for each state of the market
high_vol_avg_returns = pd.DataFrame(high_vol_stats.loc["mean"])
high_vol_avg_returns["state"] = states_dict[0]
```

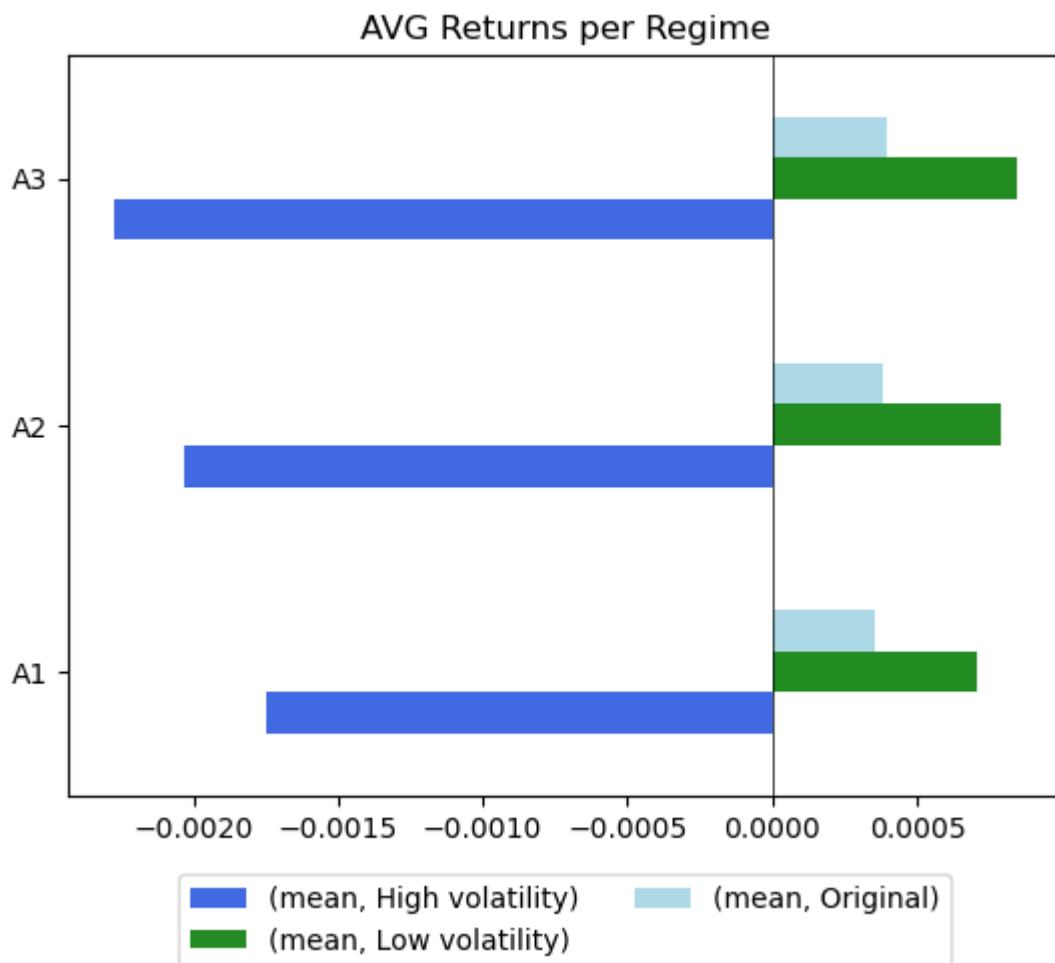
```
# Separate the mean into a different DataFrame for each state of the market
original_avg_returns = pd.DataFrame(original_stats.loc["mean"])
original_avg_returns["state"] = states_dict[2]

# Concatenate in the same table for plotting
avg_returns_df = pd.concat([low_vol_avg_returns, high_vol_avg_returns, original_avg_returns])
avg_returns_df.drop("state", inplace = True)

# Plot the graph
ax = avg_returns_df.pivot(columns='state').plot.barh(color=['royalblue', 'forestgreen'])
plt.title('AVG Returns per Regime')

# Add a Line at 0 and place the legend outside of the graph
ax.axvline(0, color='black', linewidth=0.5)
ax.legend(loc='lower center', bbox_to_anchor=(0.5, -0.25), ncol=len(avg_returns_df.columns))

# Add a Line at each maximum value
plt.show()
```



In [26]:

```
# Separate the standard deviation into a different DataFrame for each state of the market
low_vol_avg_returns = pd.DataFrame(low_vol_stats.loc["std"])
low_vol_avg_returns["state"] = states_dict[1]

# Separate the standard deviation into a different DataFrame for each state of the market
high_vol_avg_returns = pd.DataFrame(high_vol_stats.loc["std"])
high_vol_avg_returns["state"] = states_dict[0]

# Separate the standard deviation into a different DataFrame for each state of the market
```

```

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

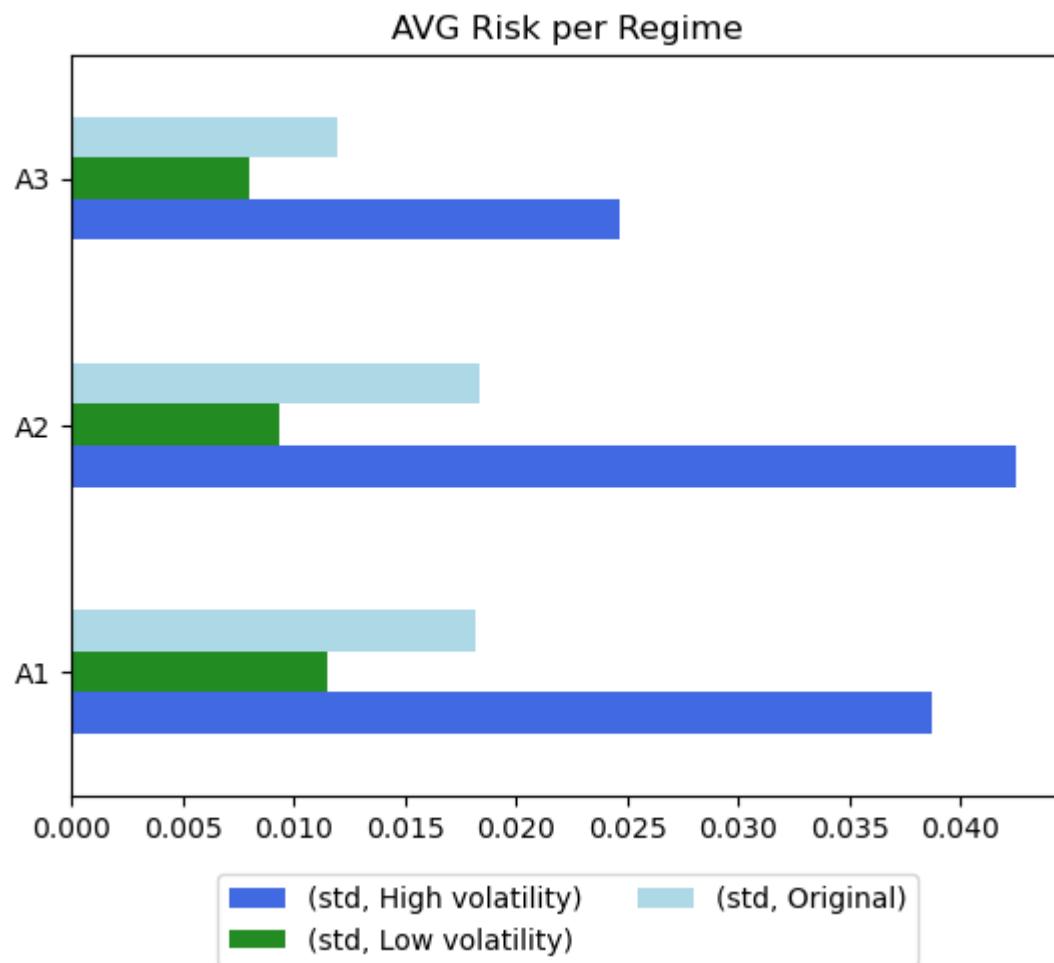
# Concatenate in the same table for plotting
avg_returns_df = pd.concat([low_vol_avg_returns, high_vol_avg_returns, original_avg_returns])
avg_returns_df.drop("state", inplace = True)

# Plot the graph
ax = avg_returns_df.pivot(columns='state').plot.barh(color=['royalblue', 'forestgreen'])
plt.title('AVG Risk per Regime')

# Add a Line at 0 and place the legend outside of the graph
ax.axvline(0, color='black', linewidth=0.5)
ax.legend(loc='lower center', bbox_to_anchor=(0.5, -0.25), ncol=len(avg_returns_df.columns))

# Add a Line at each maximum value
plt.show()

```



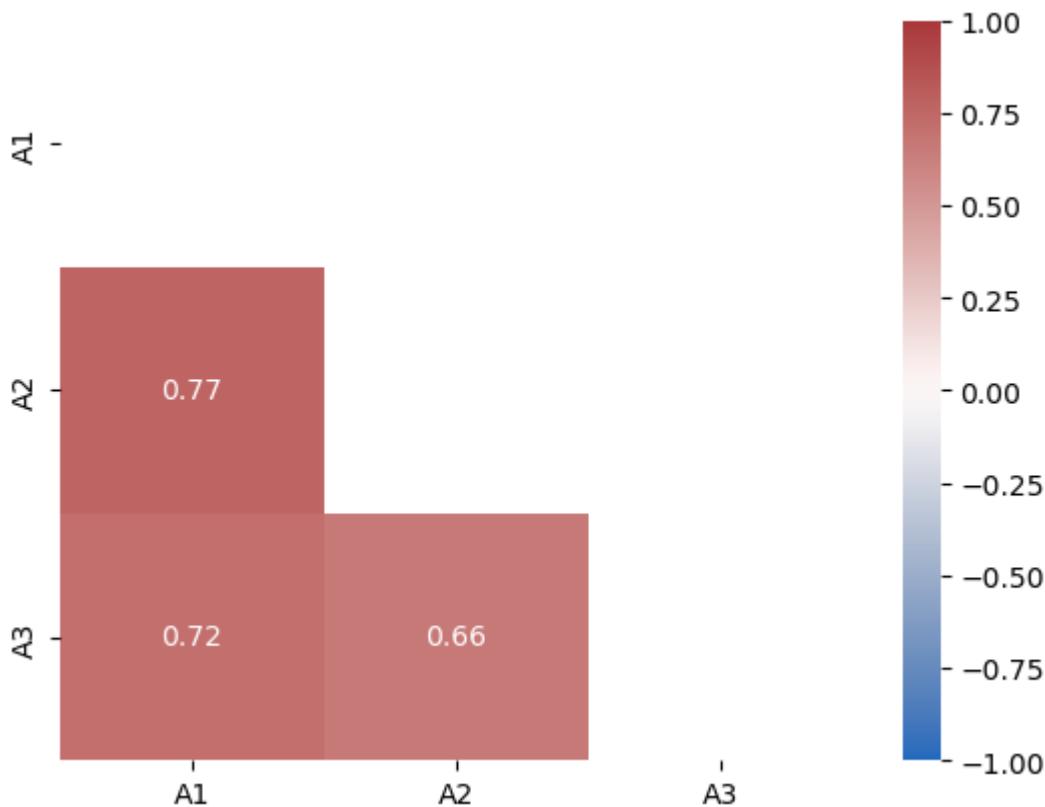
In [27]:

```

# Calculating correlation between Assets
corr_matrix = high_vol_returns_df[['A1', "A2", "A3"]].corr()
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

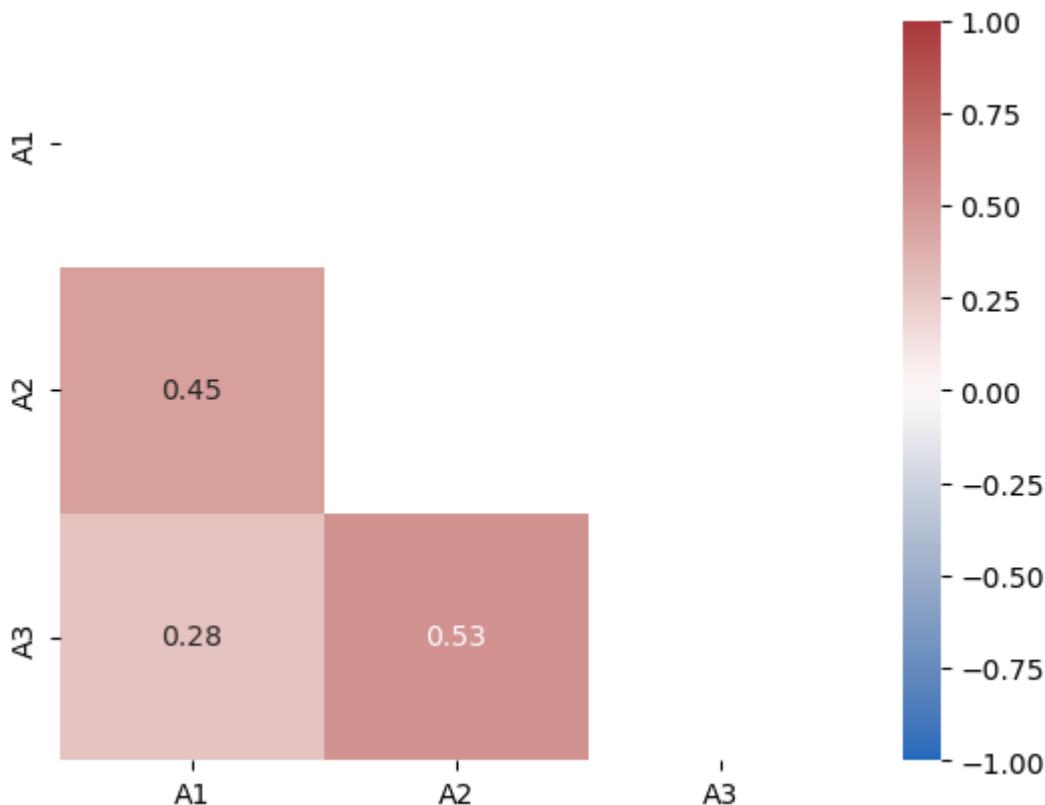
# Plot the correlation matrix as a heatmap
sns.heatmap(corr_matrix, annot=True, vmin=-1, vmax=1, cmap="vlag", mask=mask)
plt.show()

```



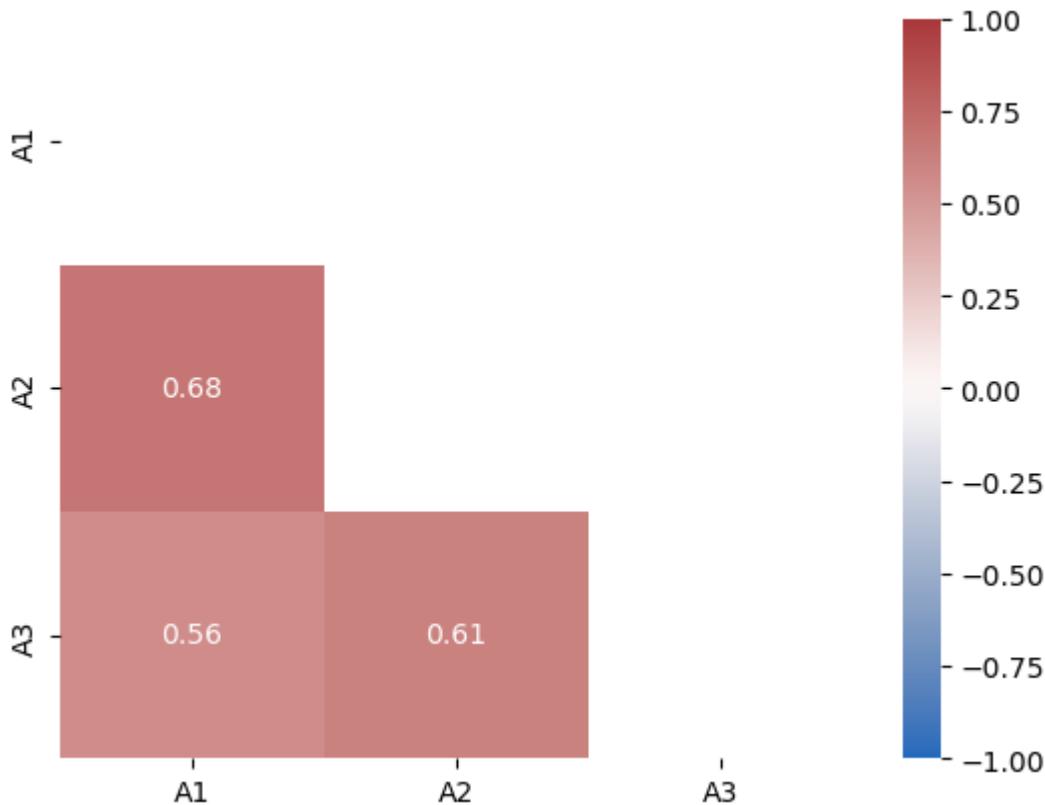
```
In [28]: # Calculating correlation between Assets
corr_matrix = low_vol_returns_df[["A1", "A2", "A3"]].corr()
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# 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]: # Calculating correlation between Assets
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()
```



## Portfolio Creation and Testing

### Both states combined

```
In [30]: # Create Dataframes containing the prices and the returns of each asset and in each state
# Low Volatility state, prices
low_vol_df_s = low_vol_df.copy()
low_vol_df_s = low_vol_df_s[['A1','A2','A3']]

# Low Volatility state, returns
low_vol_returns_df_s = low_vol_returns_df.copy()
low_vol_returns_df_s = low_vol_returns_df_s[['A1','A2','A3']]

# high Volatility state, prices
high_vol_df_s = high_vol_df.copy()
high_vol_df_s = high_vol_df_s[['A1','A2','A3']]

# high Volatility state, returns
high_vol_returns_df_s = high_vol_returns_df.copy()
high_vol_returns_df_s = high_vol_returns_df_s[['A1','A2','A3']]

# Original, both states combined, prices
states_df_s = states_df.copy()
states_df_s = states_df_s[['A1','A2','A3']]

# Original, both states combined, returns
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
```

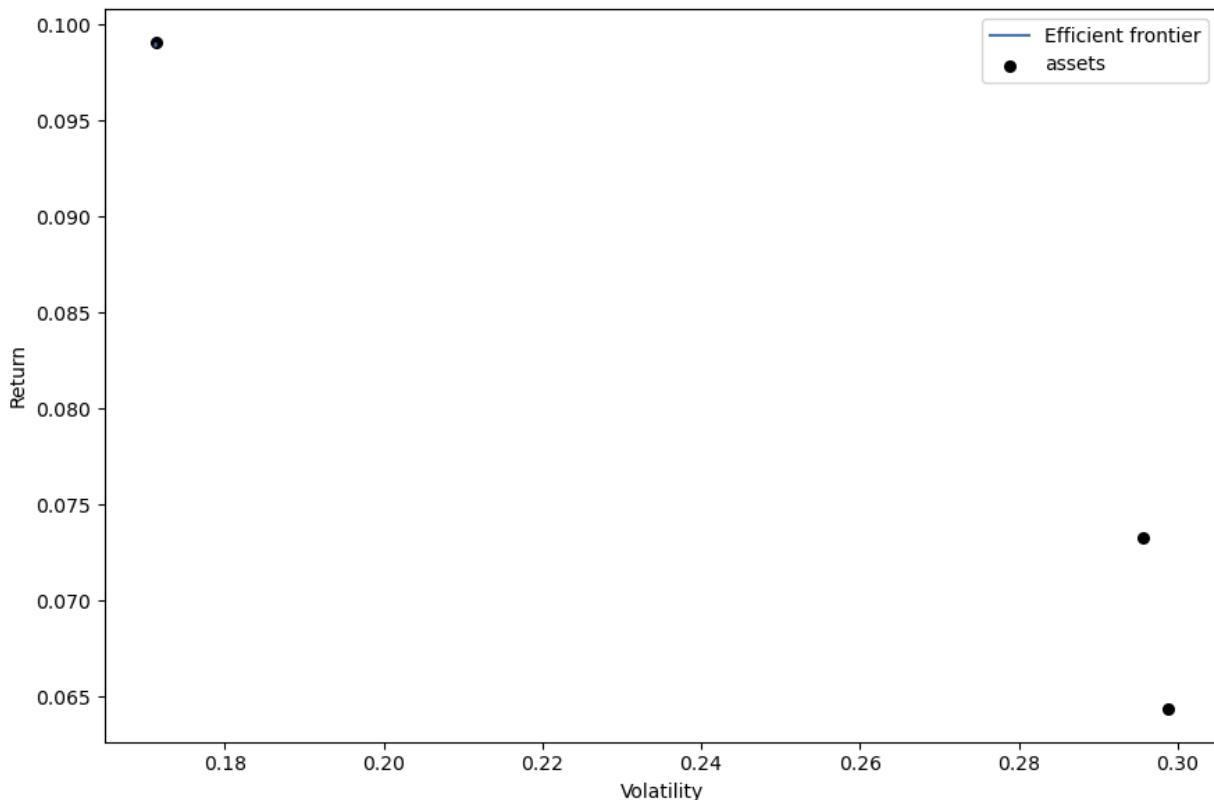
```
In [33]: # historical covariance matrix
S = risk_models.sample_cov(states_train)
S
```

```
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://pyportfoliopp.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()
```

```
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))
sharps = rets / stds
sharps
```

```
Out[39]: array([0.41725959, 0.34277297, 0.33911361, ..., 0.31644514, 0.29831892,
 0.37016377])
```

```
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)])
```

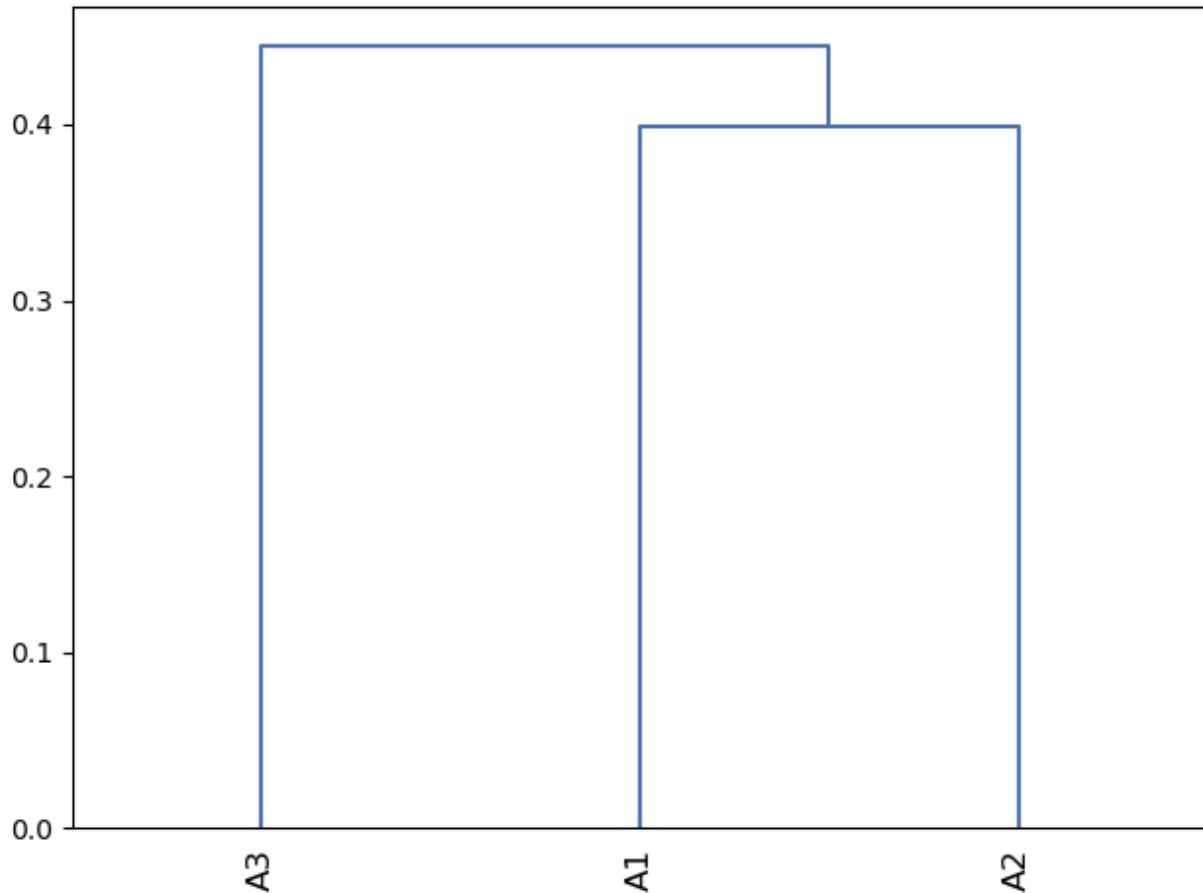
In [41]: # Calculating Metrics

```
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]: # Plotting both Portfolio compositions

```
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")
```

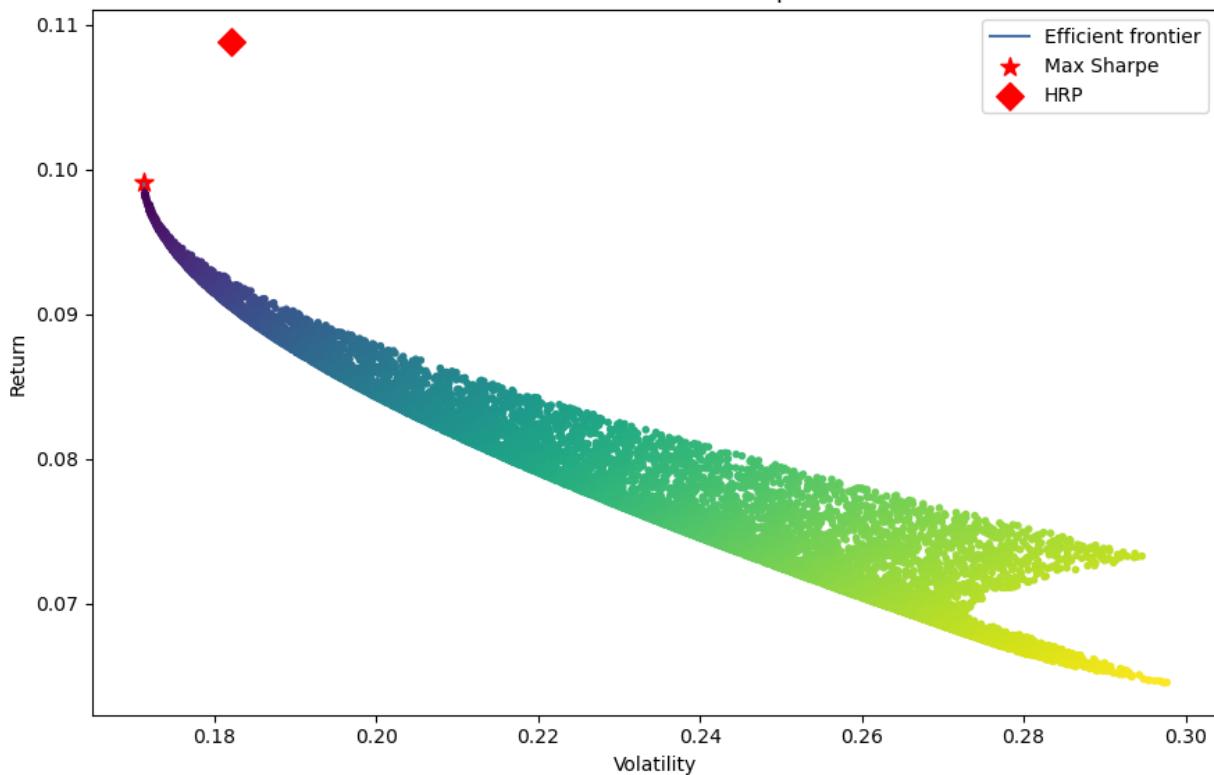
```
ax.set_title("Efficient Frontier with random portfolios")
```

```
ax.legend()
```

```
plt.tight_layout()
```

```
plt.show()
```

## Efficient Frontier with random portfolios



```
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
```

Out[46]:

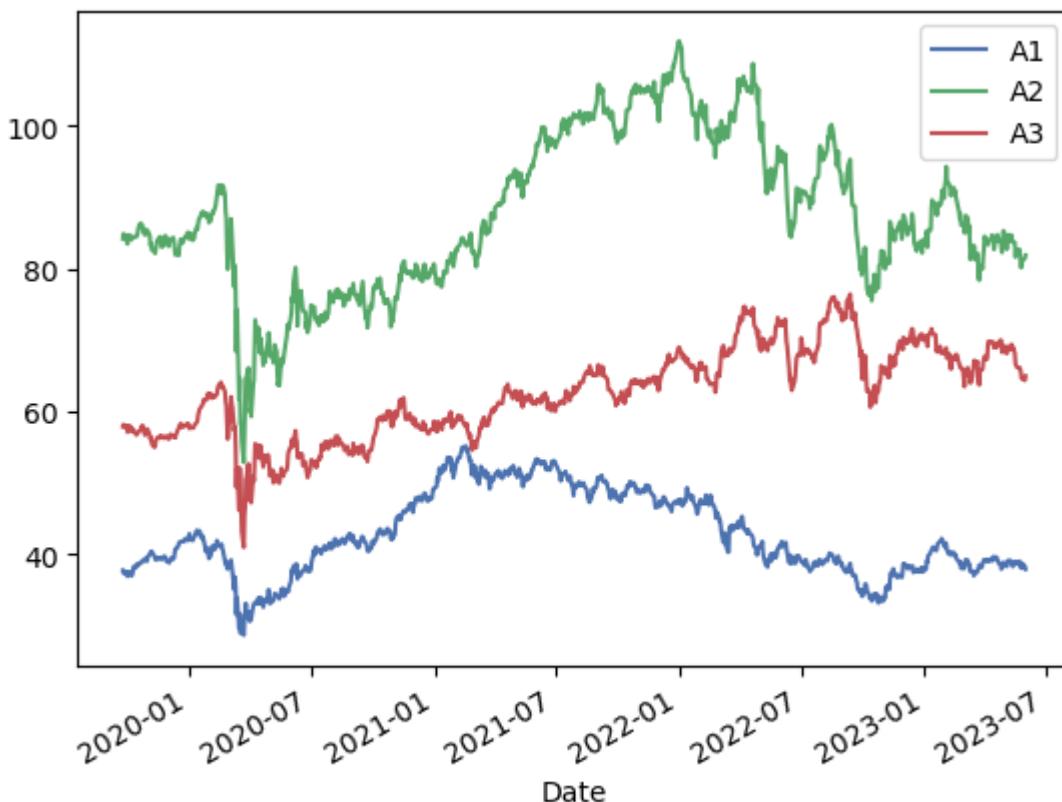
port\_tangent port\_hrp

Date	port_tangent	port_hrp
2019-09-25	-0.000619	-0.000438
2019-09-26	0.004642	0.004469
2019-09-27	0.001083	-0.000766
2019-09-30	0.001702	0.001005
2019-10-01	-0.000619	-0.003168
...	...	...
2023-05-24	0.131302	0.107121
2023-05-25	0.116096	0.096150
2023-05-26	0.116096	0.100685
2023-05-30	0.111776	0.096529
2023-05-31	0.121971	0.103128

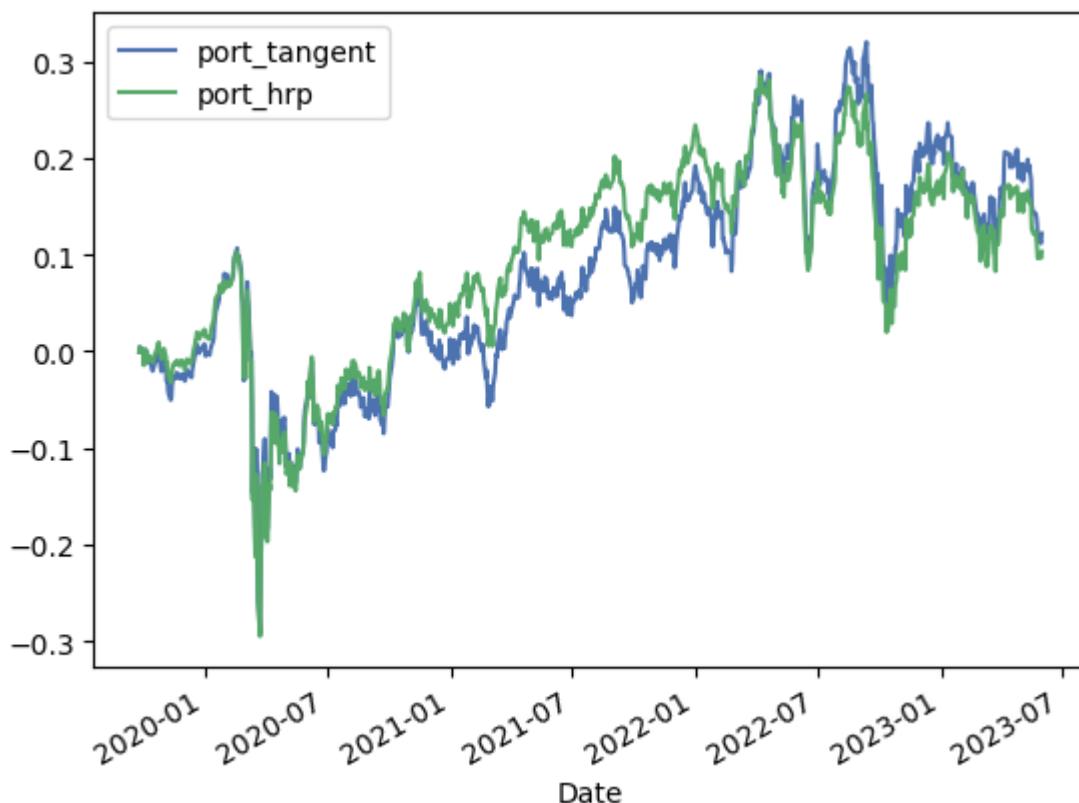
927 rows × 2 columns

In [47]: states\_test.plot()

Out[47]: &lt;AxesSubplot:xlabel='Date'&gt;

In [48]: # out-of-sample performance  
port\_equity.plot()

Out[48]: &lt;AxesSubplot:xlabel='Date'&gt;

In [49]: # out-of-sample volatilities  
port\_equity.std()Out[49]: port\_tangent 0.107861  
port\_hrp 0.100904  
dtype: float64

## Low Volatility State

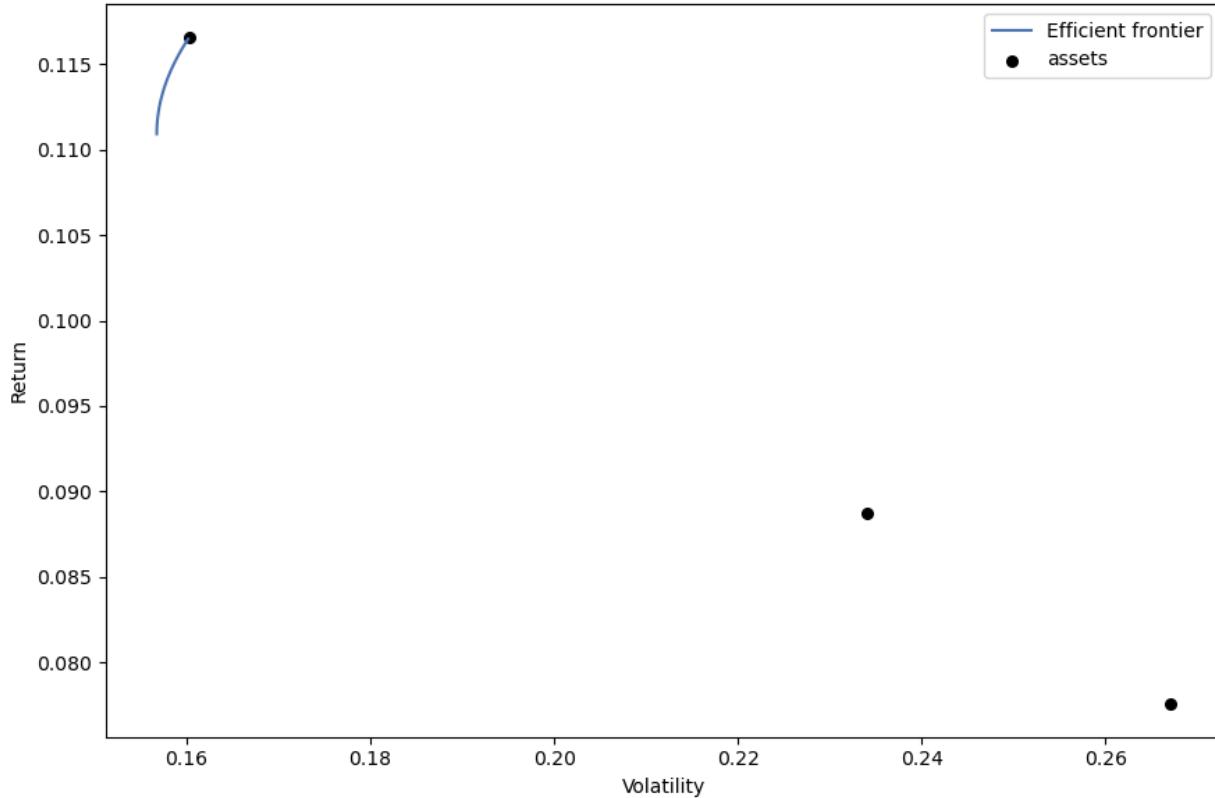
In [50]: # average historical returns  
mu = expected\_returns.mean\_historical\_return(low\_vol\_train)  
mu.sort\_values(ascending=False)Out[50]: A3 0.116555  
A2 0.088726  
A1 0.077613  
dtype: float64In [51]: # historical covariance matrix  
S = risk\_models.sample\_cov(low\_vol\_train)  
SOut[51]: 

	A1	A2	A3
A1	0.071284	0.040395	0.017892
A2	0.040395	0.054772	0.020280
A3	0.017892	0.020280	0.025695

```
In [52]: # find efficient frontier
ef = EfficientFrontier(mu, S)
```

```
In [53]: # save object copies for further calculations and plotting (auxiliary step)
ef_cp = ef.deepcopy()
ef_tangent = ef.deepcopy()
```

```
In [54]: # plot efficient frontier: see PyPortfolioOpt docs
# https://pyportfoliopp.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 [55]: # find the tangency (max. Sharpe ratio) portfolio
ef_tangent.max_sharpe()
ret_tangent, std_tangent, _ = ef_tangent.portfolio_performance(verbose=True)
```

Expected annual return: 11.7%

Annual volatility: 16.0%

Sharpe Ratio: 0.60

```
In [56]: # tangency portfolio weights
tangent_weights = ef_tangent.clean_weights()
tangent_weights
```

Out[56]: OrderedDict([('A1', 0.0), ('A2', 0.0), ('A3', 1.0)])

```
In [57]: # 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))
```

```
sharpe = rets / stds
sharpe

Out[57]: array([0.45115376, 0.40700269, 0.44006153, ..., 0.46581966, 0.31518937,
   0.51648571])
```

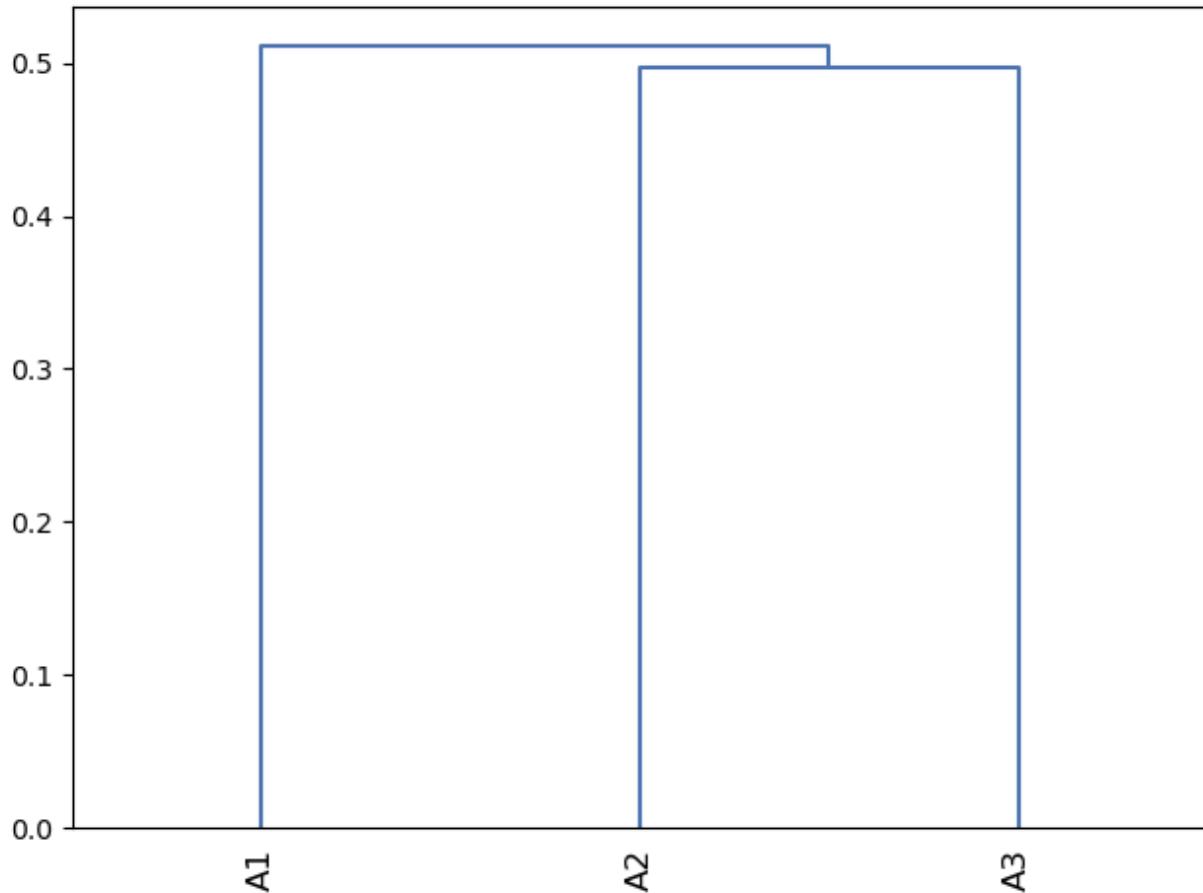
```
In [58]: # Hierarchical Risk Parity (HRP) portfolio
hrp = HRPOpt(low_vol_returns_train)
hrp_weights = hrp.optimize()
hrp_weights
```

```
Out[58]: OrderedDict([('A1', 0.27260424258482474),
 ('A2', 0.299567256653504),
 ('A3', 0.4278285007616713)])
```

```
In [59]: ret_hrp, std_hrp, _ = hrp.portfolio_performance(verbose=True)
```

Expected annual return: 20.5%  
 Annual volatility: 11.5%  
 Sharpe Ratio: 1.60

```
In [60]: # Hierarchical Clustering dendrogram
plotting.plot_dendrogram(hrp, showfig = True)
```



```
Out[60]: <AxesSubplot:>
```

```
In [61]: # Plotting both Portfolio compositions
fig, ax = plt.subplots(figsize=(9,6))
plotting.plot_efficient_frontier(ef_cp, ax=ax, show_assets=False)

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

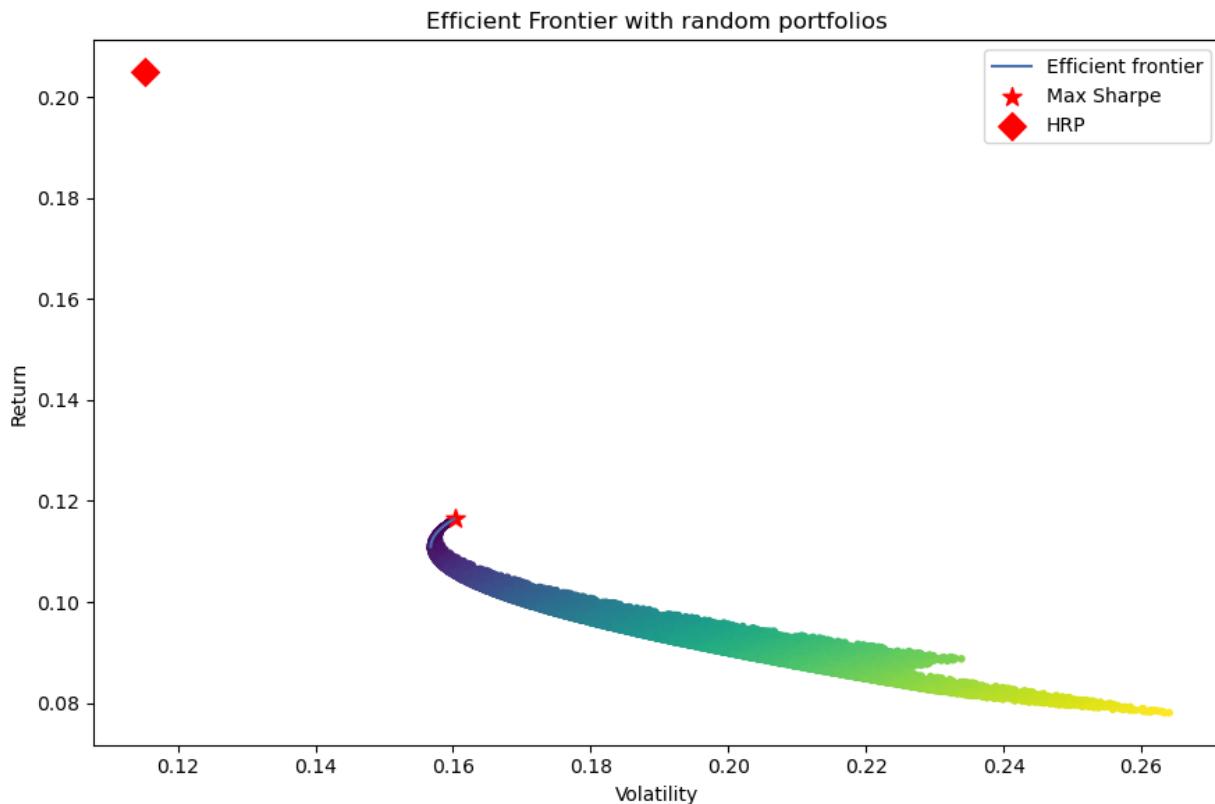
```

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()

```



In [62]:

```

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

```

In [63]:

```

low_vol_returns_test['port_hrp'] = 0
for ticker, weight in hrp_weights.items():
    low_vol_returns_test['port_hrp'] += low_vol_returns_test[ticker]*weight

```

In [64]:

```

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

```

Out[64]:

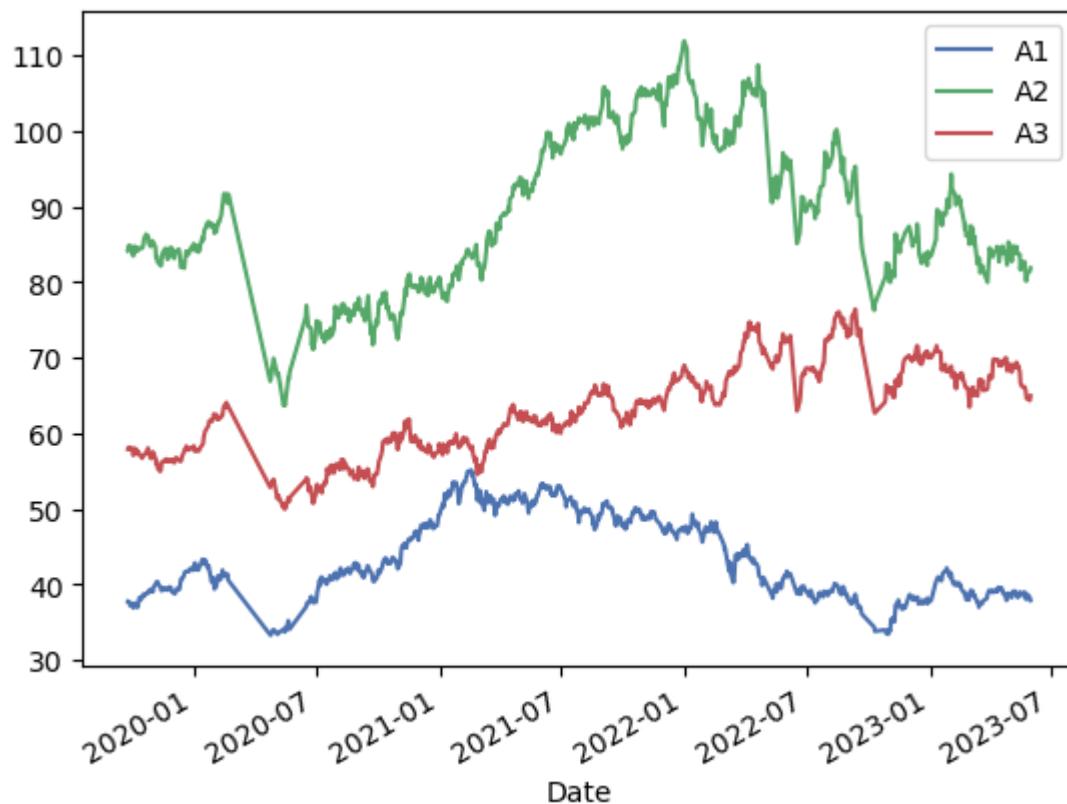
port\_tangent port\_hrp

Date	port_tangent	port_hrp
2019-09-24	0.011741	0.001770
2019-09-25	0.011114	0.001532
2019-09-26	0.016437	0.006173
2019-09-27	0.012837	-0.000716
2019-09-30	0.013463	0.002215
...	...	...
2023-05-24	0.882769	0.698477
2023-05-25	0.857462	0.687808
2023-05-26	0.857462	0.701812
2023-05-30	0.850273	0.695832
2023-05-31	0.867240	0.700839

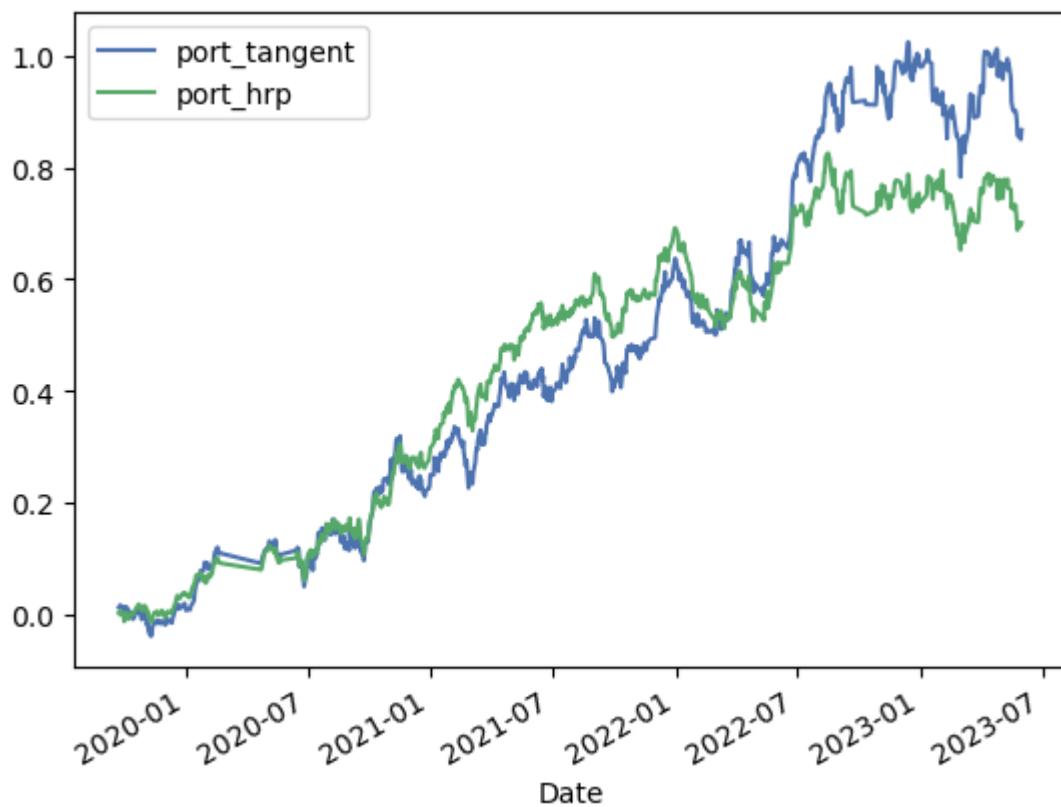
794 rows × 2 columns

In [65]: low\_vol\_test.plot()

Out[65]: &lt;AxesSubplot:xlabel='Date'&gt;

In [66]: # out-of-sample performance  
port\_equity.plot()

Out[66]: &lt;AxesSubplot:xlabel='Date'&gt;

In [67]: # out-of-sample volatilities  
port\_equity.std()Out[67]: port\_tangent 0.317391  
port\_hrp 0.257587  
dtype: float64

## High Volatility State

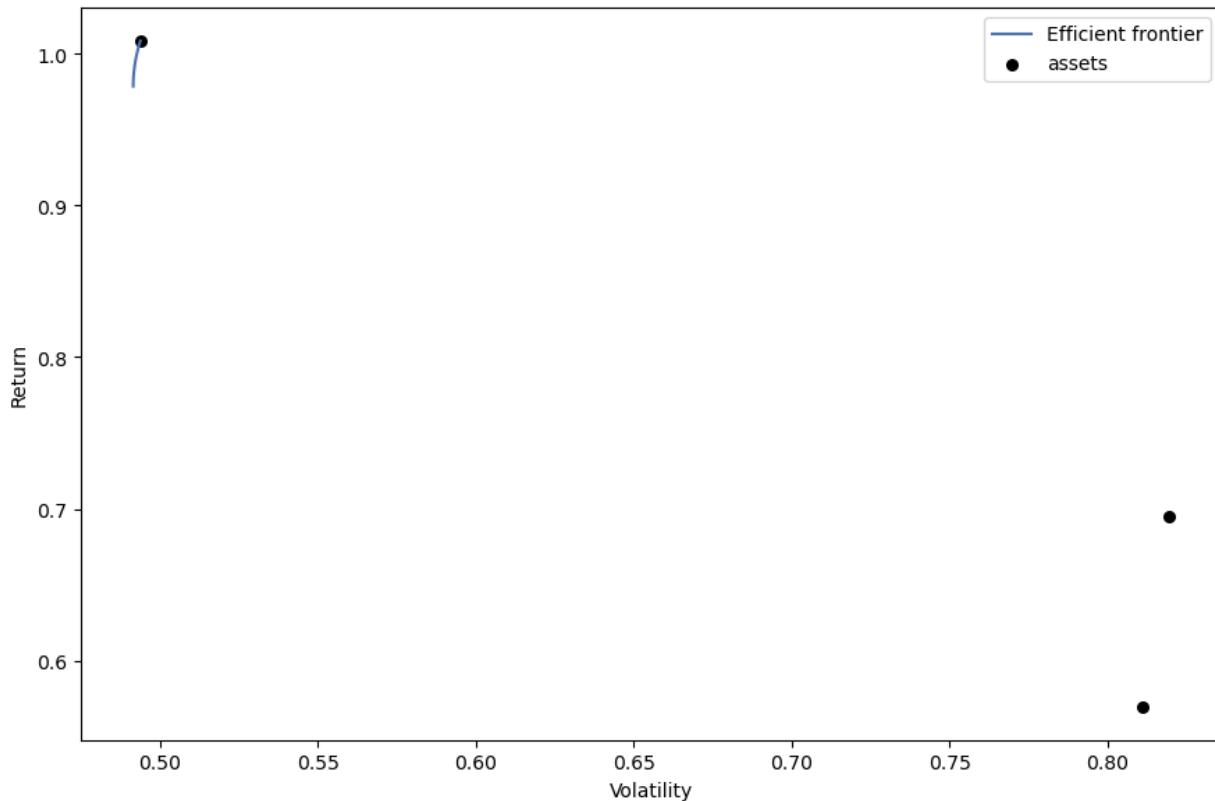
In [68]: # average historical returns  
mu = expected\_returns.mean\_historical\_return(high\_vol\_train)  
mu.sort\_values(ascending=False)Out[68]: A3 1.008731  
A2 0.694519  
A1 0.569492  
dtype: float64In [69]: # historical covariance matrix  
S = risk\_models.sample\_cov(high\_vol\_train)  
S

Out[69]:

```
In [70]: # find efficient frontier
ef = EfficientFrontier(mu, S)
```

```
In [71]: # save object copies for further calculations and plotting (auxiliary step)
ef_cp = ef.deepcopy()
ef_tangent = ef.deepcopy()
```

```
In [72]: # plot efficient frontier: see PyPortfolioOpt docs
# https://pyportfoliopp.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 [73]: # find the tangency (max. Sharpe ratio) portfolio
ef_tangent.max_sharpe()
ret_tangent, std_tangent, _ = ef_tangent.portfolio_performance(verbose=True)
```

Expected annual return: 100.9%

Annual volatility: 49.4%

Sharpe Ratio: 2.00

```
In [74]: # tangency portfolio weights
tangent_weights = ef_tangent.clean_weights()
tangent_weights
```

```
Out[74]: OrderedDict([('A1', 0.0), ('A2', 0.0), ('A3', 1.0)])
```

```
In [75]: # 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))
```

```
sharpe = rets / stds
sharpe

Out[75]: array([0.93120036, 0.94000871, 1.41155692, ..., 0.95558678, 0.98009046,
   0.98403315])
```

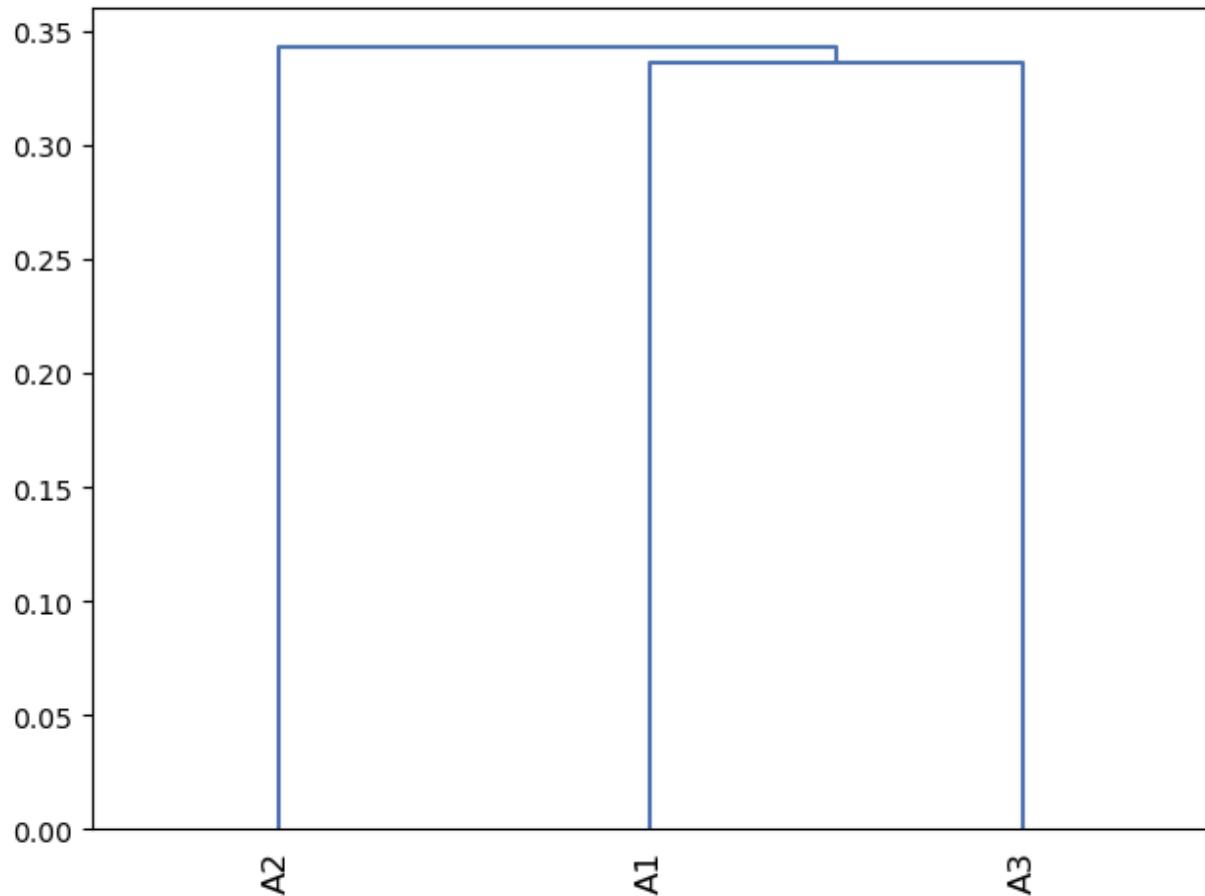
```
In [76]: # Hierarchical Risk Parity (HRP) portfolio
hrp = HRPOpt(high_vol_returns_train)
hrp_weights = hrp.optimize()
hrp_weights
```

```
Out[76]: OrderedDict([('A1', 0.1660694837609595),
 ('A2', 0.23403797056583764),
 ('A3', 0.5998925456732028)])
```

```
In [77]: ret_hrp, std_hrp, _ = hrp.portfolio_performance(verbose=True)
```

Expected annual return: -48.7%  
 Annual volatility: 42.6%  
 Sharpe Ratio: -1.19

```
In [78]: # Hierarchical Clustering dendrogram
plotting.plot_dendrogram(hrp, showfig = True)
```



```
Out[78]: <AxesSubplot:>
```

```
In [79]: # Plotting both Portfolio compositions
fig, ax = plt.subplots(figsize=(9,6))
plotting.plot_efficient_frontier(ef_cp, ax=ax, show_assets=False)

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

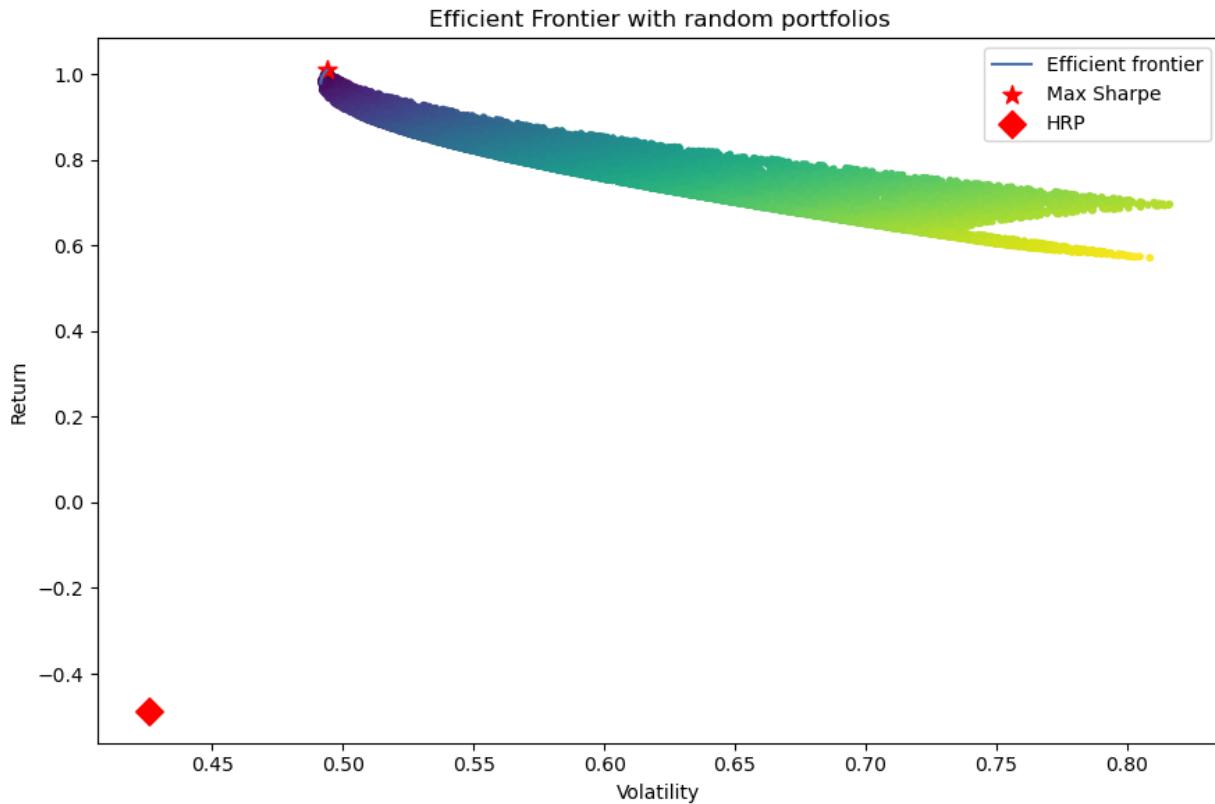
```

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()

```



```
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
```

Out[82]:

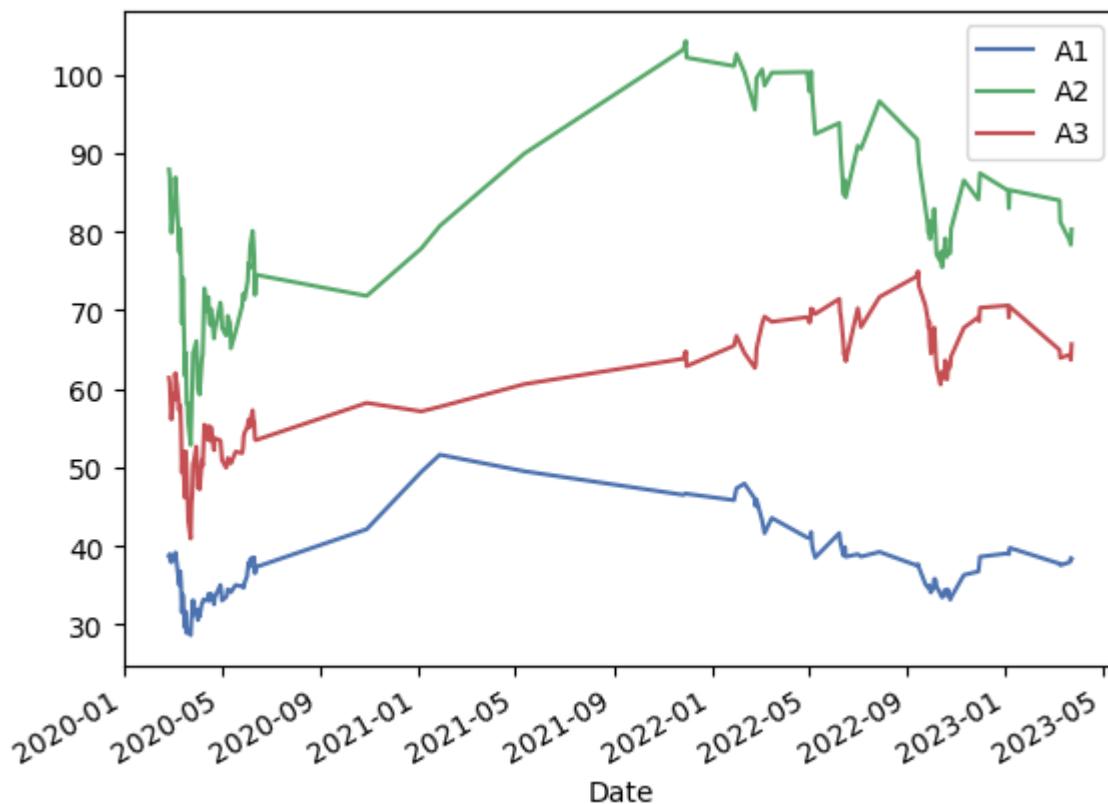
port\_tangent port\_hrp

Date	port_tangent	port_hrp
2020-02-25	-0.021004	-0.020188
2020-02-26	-0.030931	-0.027256
2020-02-27	-0.074522	-0.069314
2020-02-28	-0.105453	-0.095352
2020-03-02	-0.052798	-0.049947
...	...	...
2023-03-09	-0.375448	-0.358054
2023-03-10	-0.385277	-0.369613
2023-03-22	-0.397356	-0.381762
2023-03-23	-0.403536	-0.384999
2023-03-24	-0.384902	-0.370154

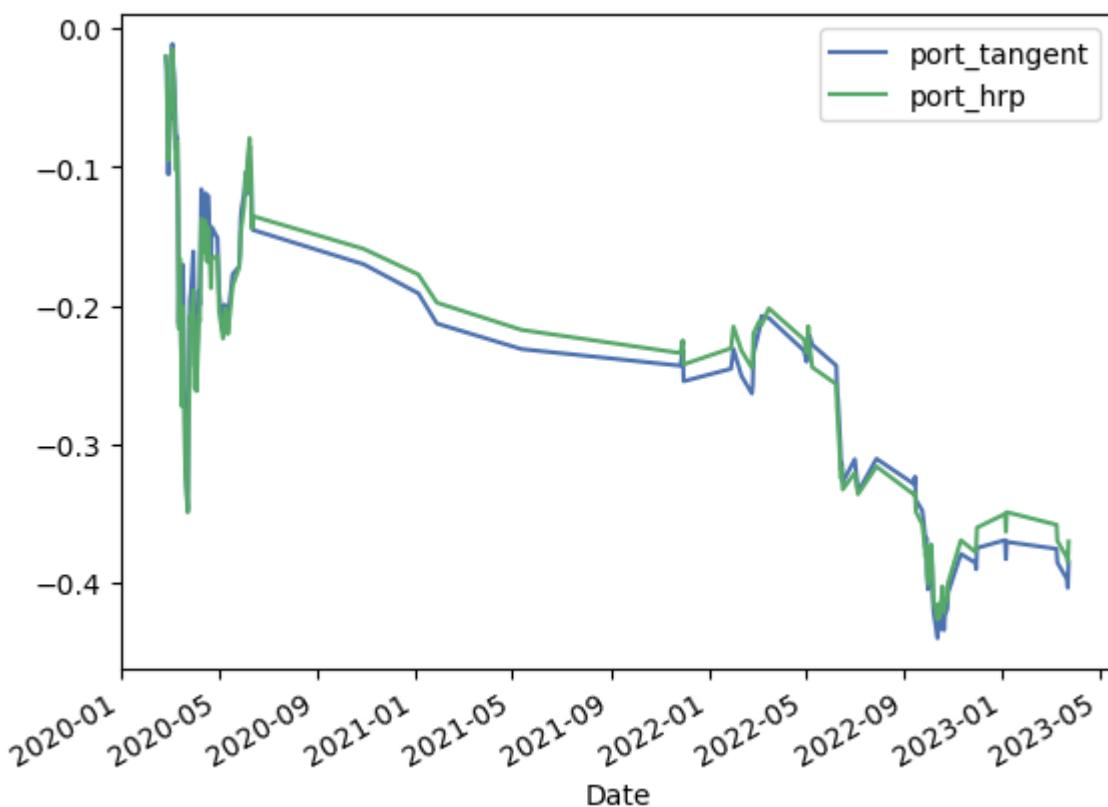
133 rows × 2 columns

In [83]: high\_vol\_test.plot()

Out[83]: &lt;AxesSubplot:xlabel='Date'&gt;

In [84]: # out-of-sample performance  
port\_equity.plot()

Out[84]: &lt;AxesSubplot:xlabel='Date'&gt;

In [85]: # out-of-sample volatilities  
port\_equity.std()Out[85]: port\_tangent 0.112450  
port\_hrp 0.108273  
dtype: float64

## Recommendations

In both Low-volatility and Original state the Maximum Sharpe portfolio outperforms the HRP portfolio strategy. Therefore, the algorithm suggests investing only in A3. However, in scenarios such the High-volatility state, the HRP portfolio strategy reveals to perform slightly better than the Maximum Sharpe strategy. In this case, the portfolio allocation should be the following: A1 16.6%, A2 23.4%, A3 59.9%

# 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

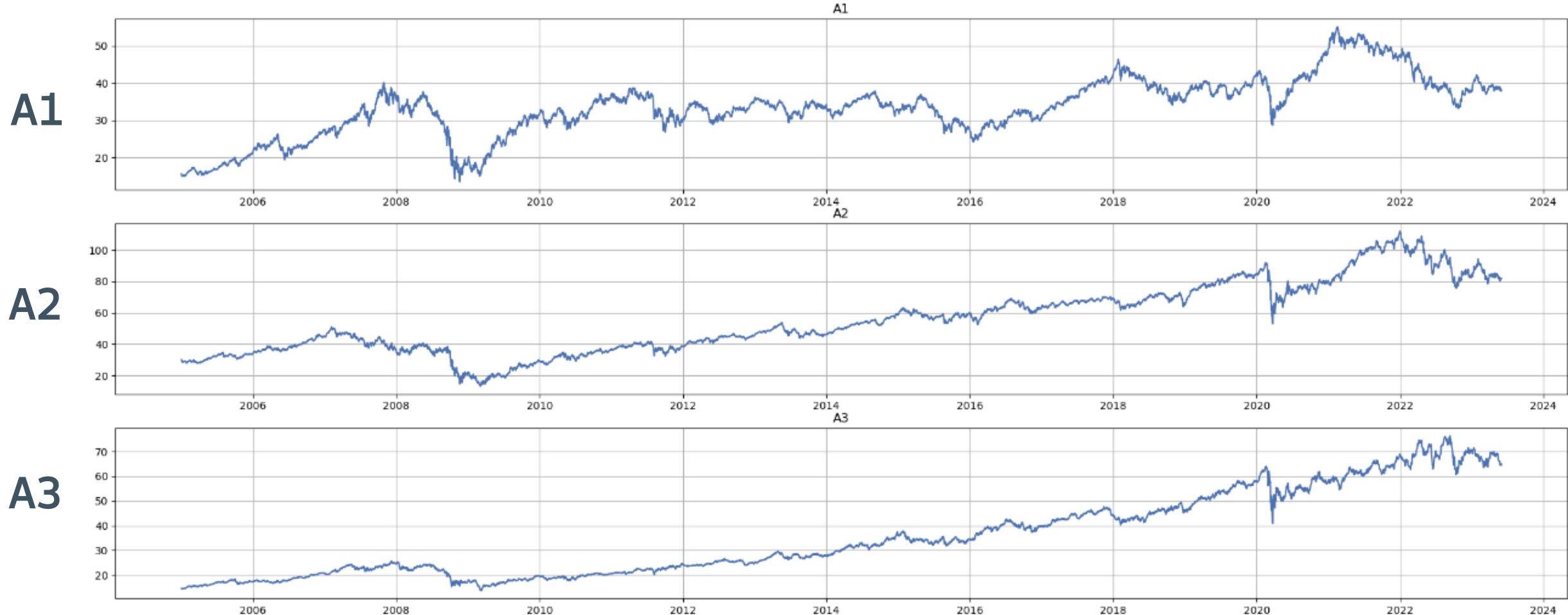
# Market Behaviour

Visualization to compare price movement over time for each asset

2007- Financial Crisis

2020-Covid-19 Pandemic

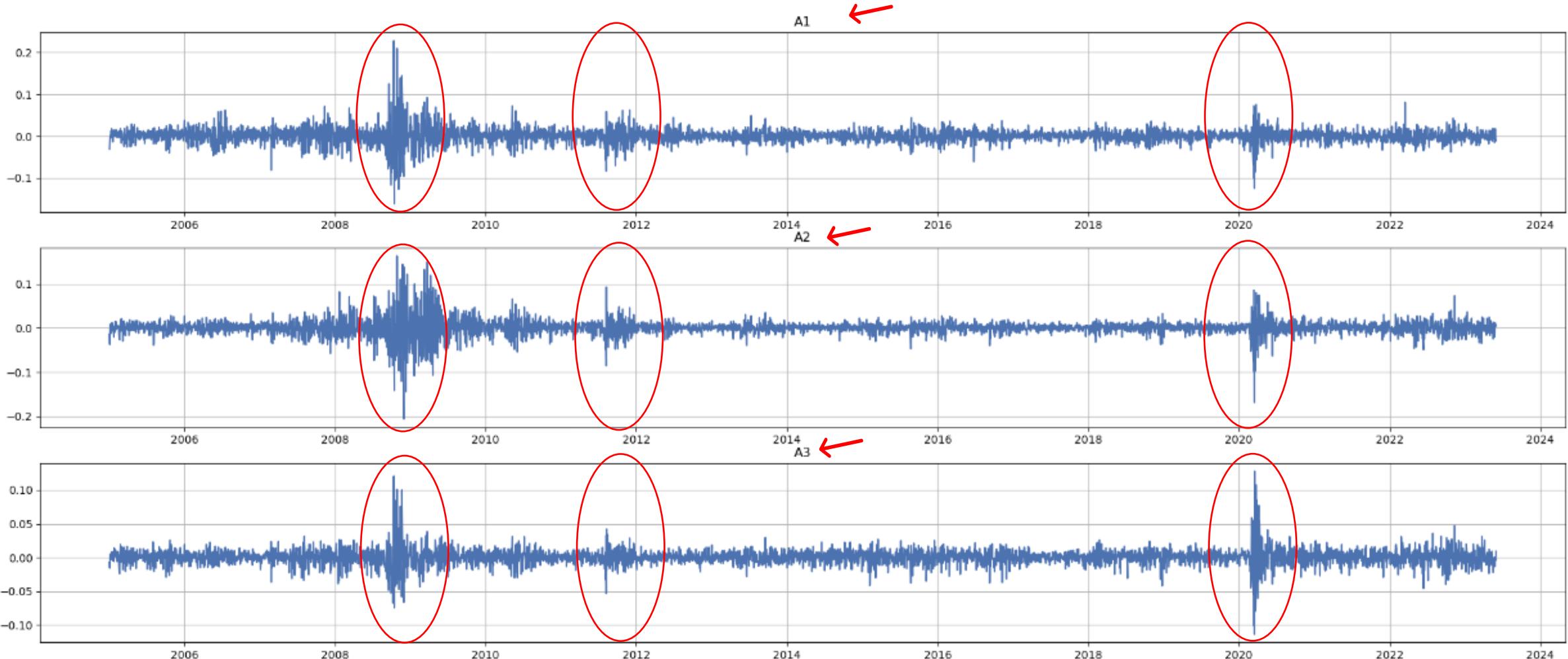
The share price of Assets 2005-2023



# 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

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

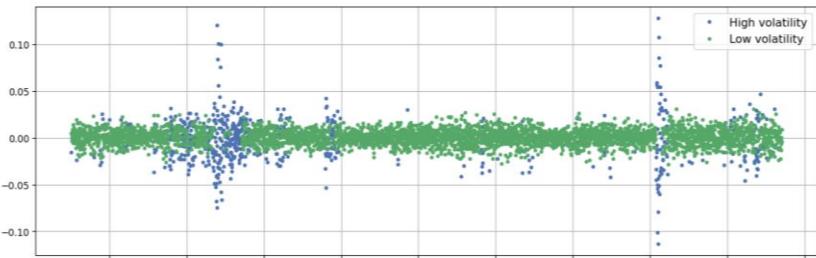
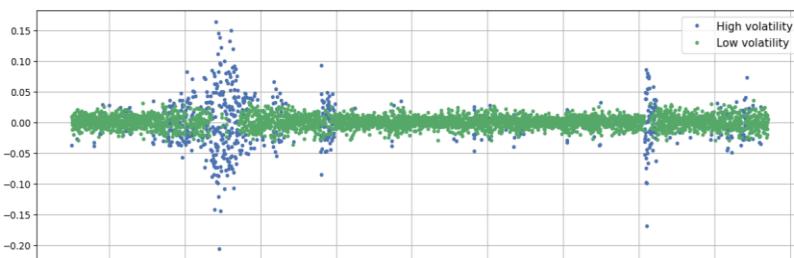
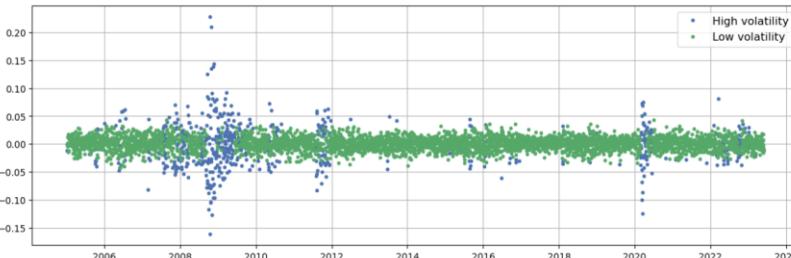
A1



A2

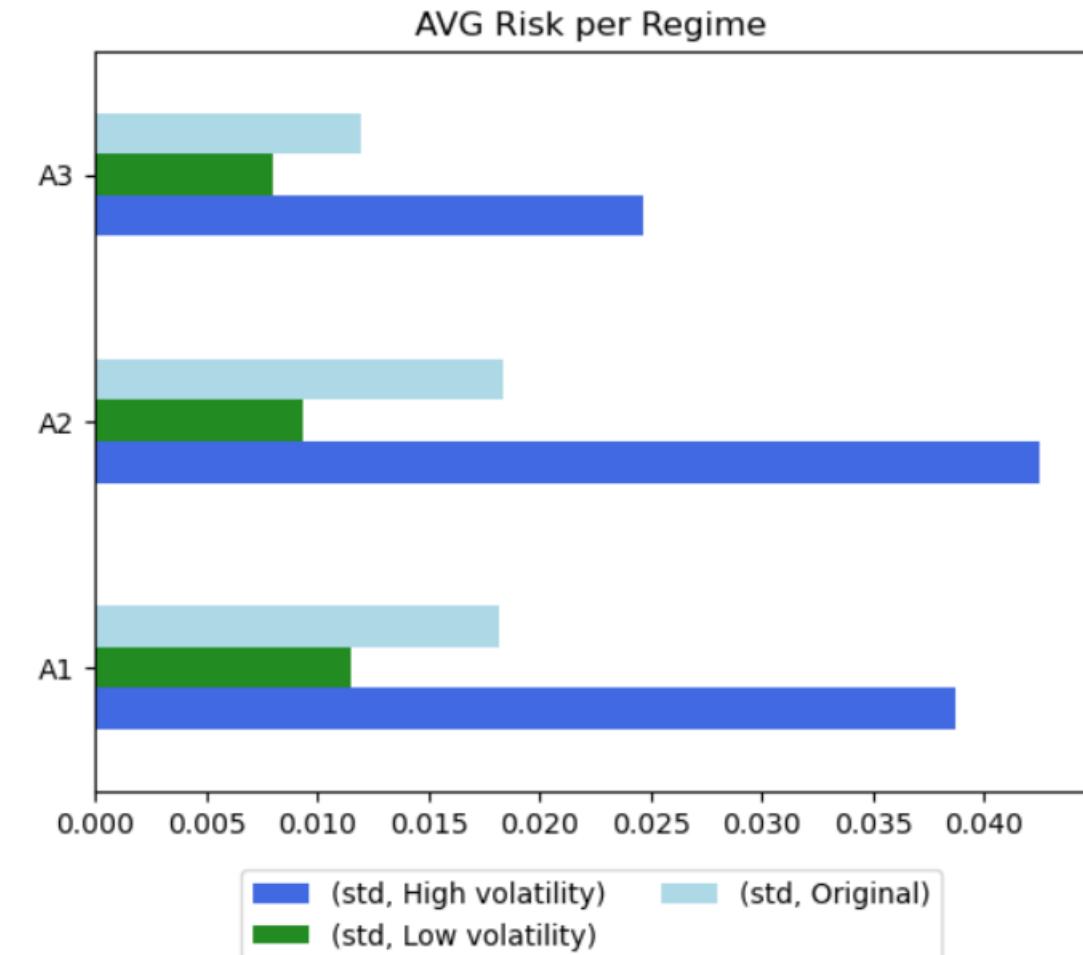
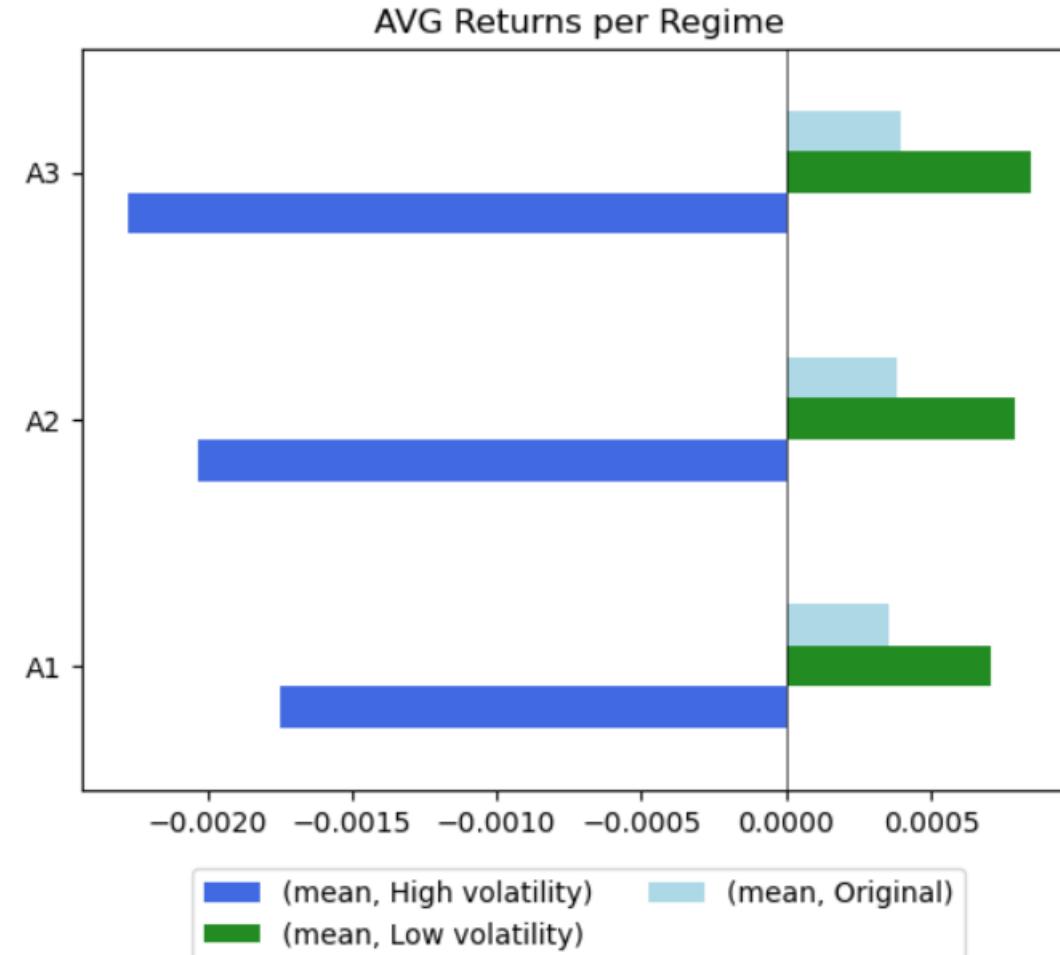


A3



# Asset returns based on Hidden States

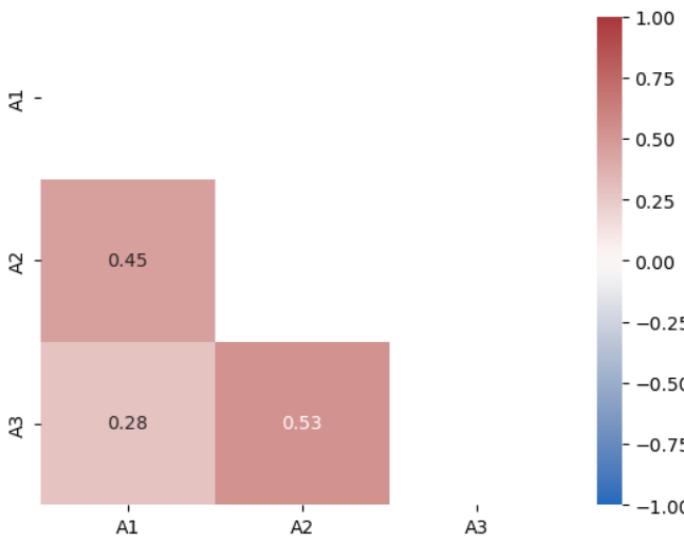
A3 is outperforming A1 and A2 in both high- and low-volatility states



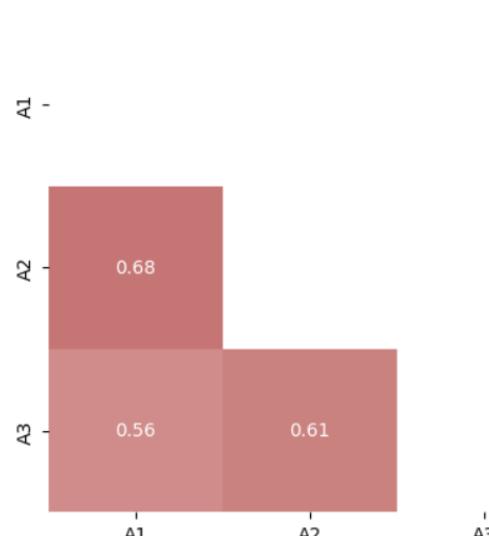
# Correlation between assets

Assets' correlation changes according to the market states

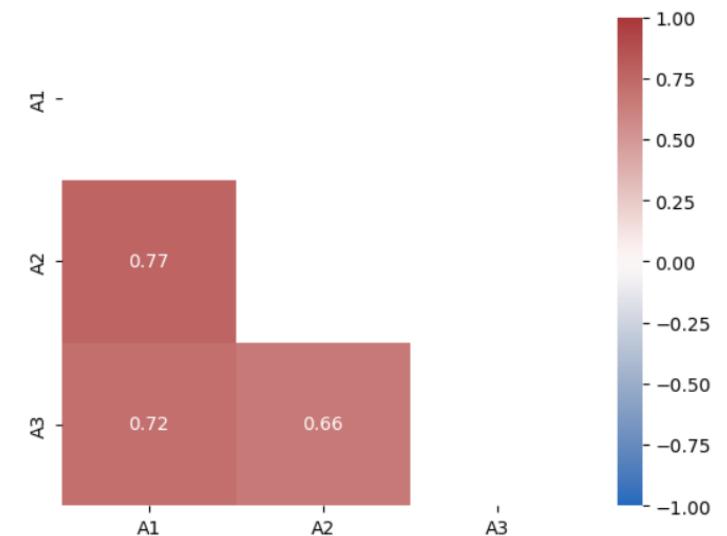
Low-Volatility State



Market State

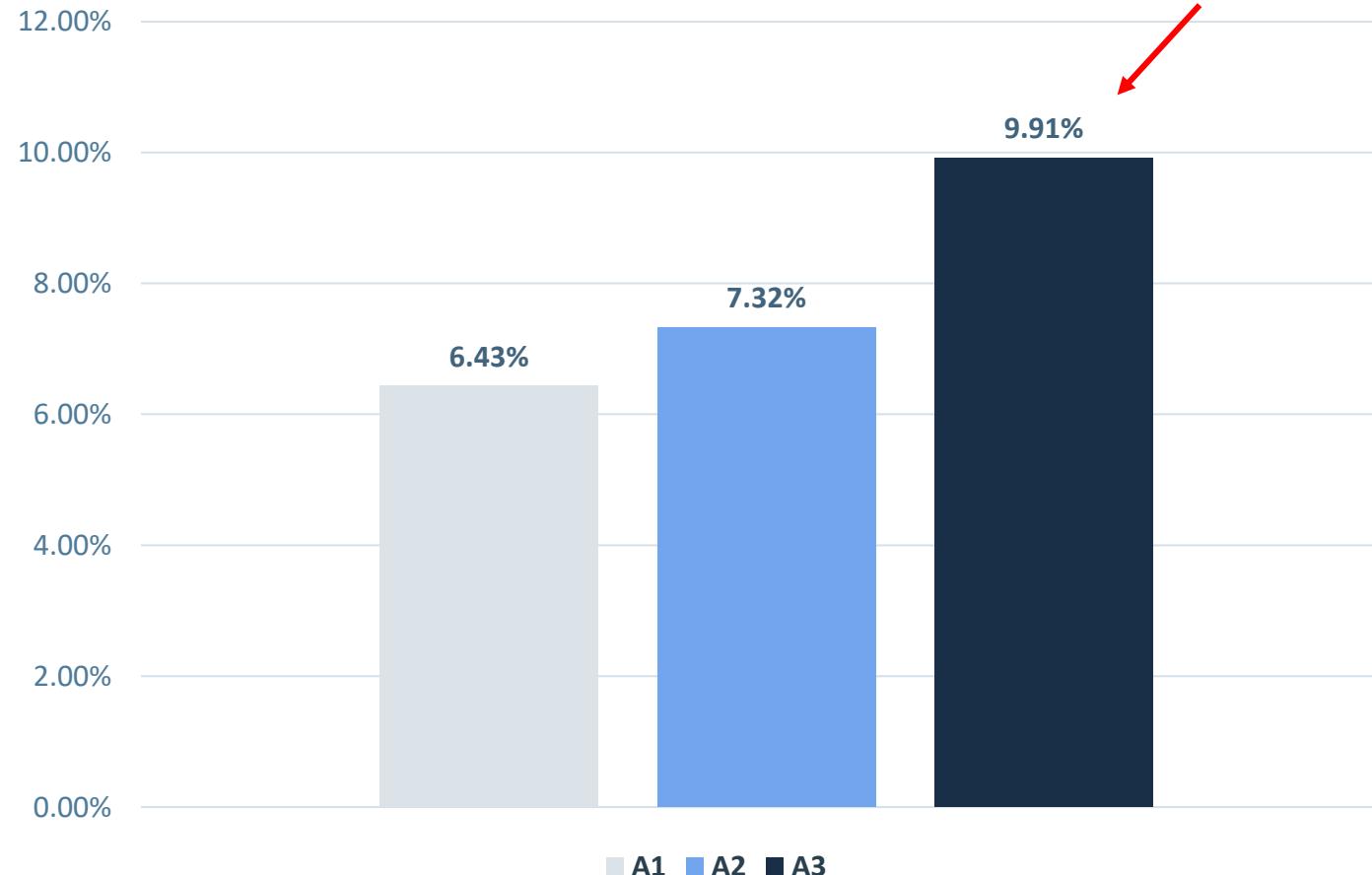


High-Volatility State



# Historical Returns

## 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

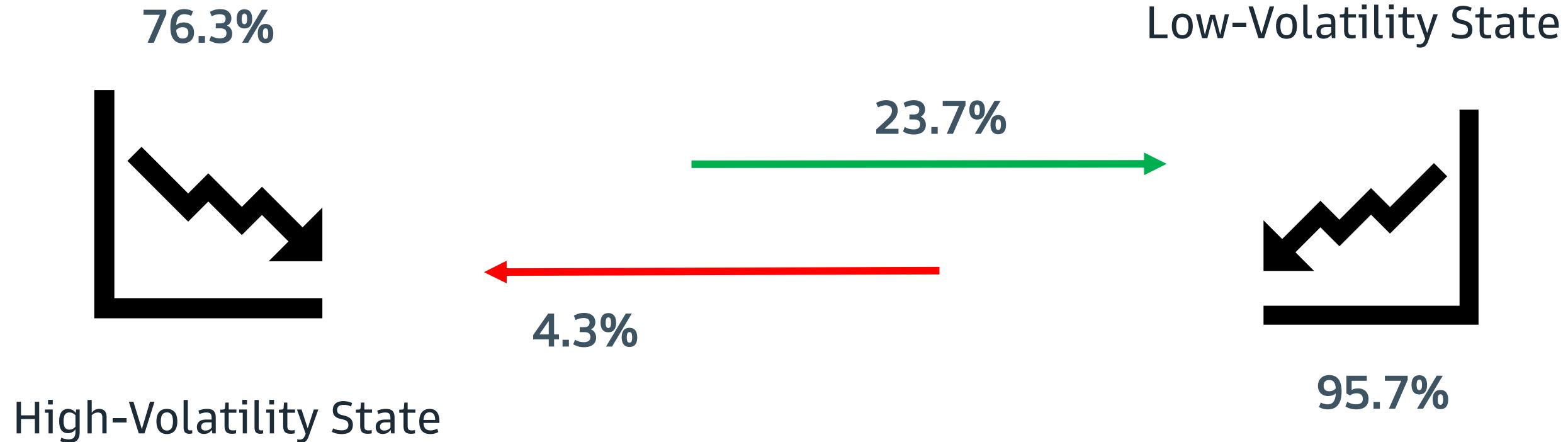
## High-Volatility State



	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

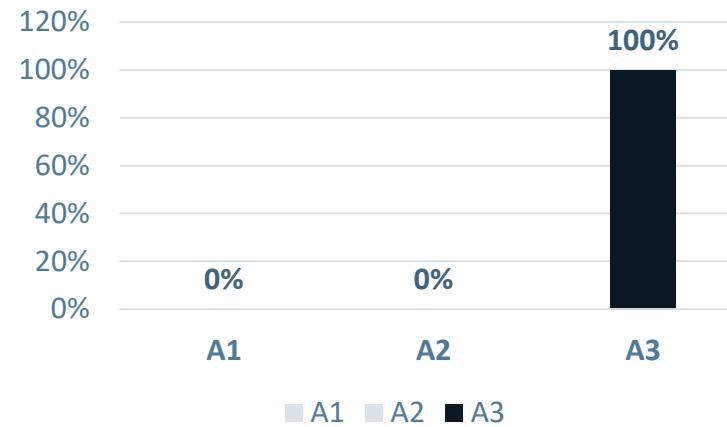


# Portfolio optimization

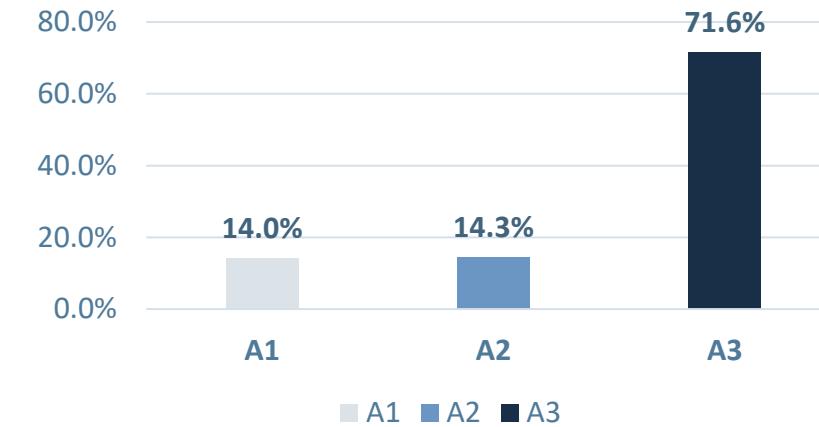
## Overall

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

Tangency portfolio weights

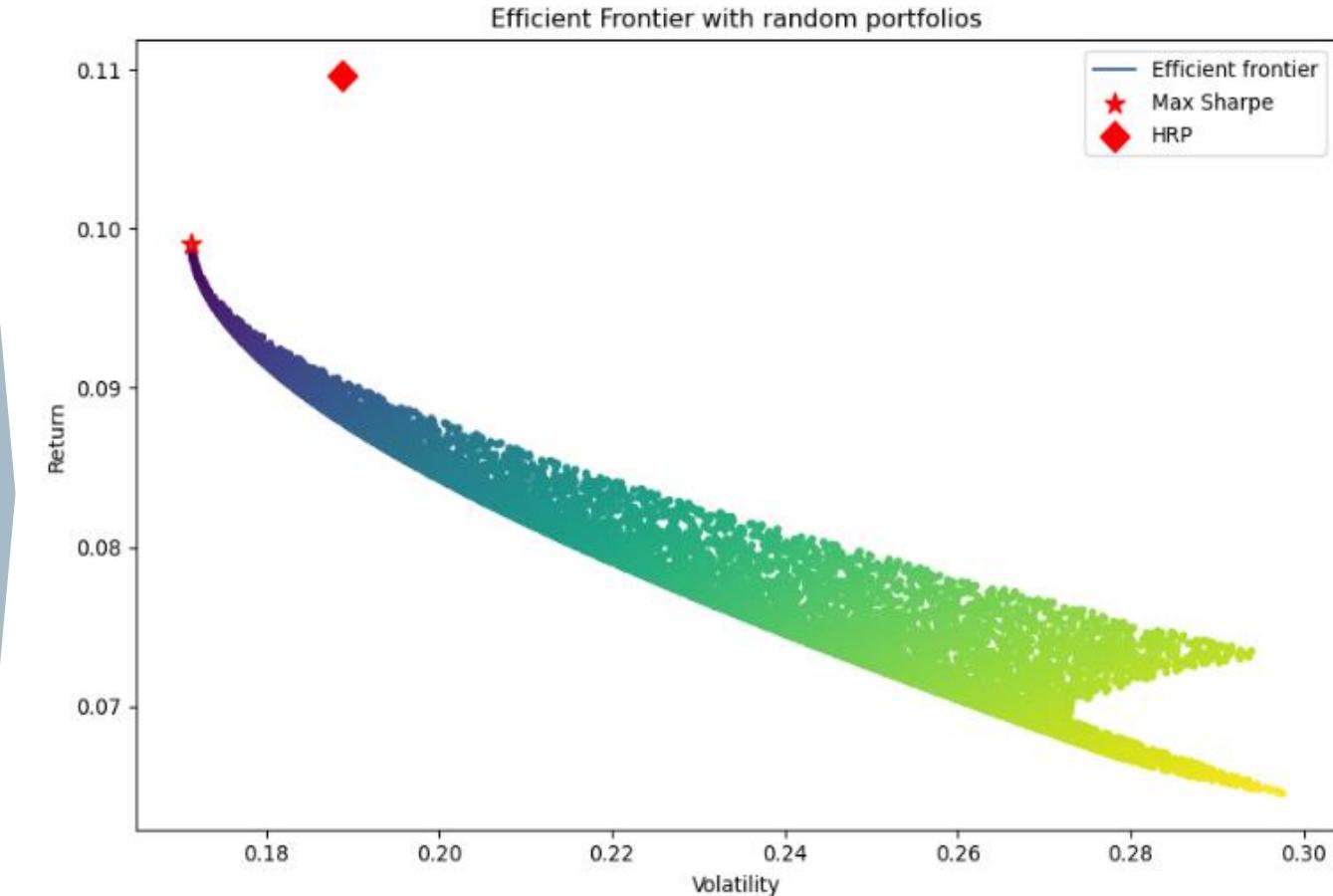
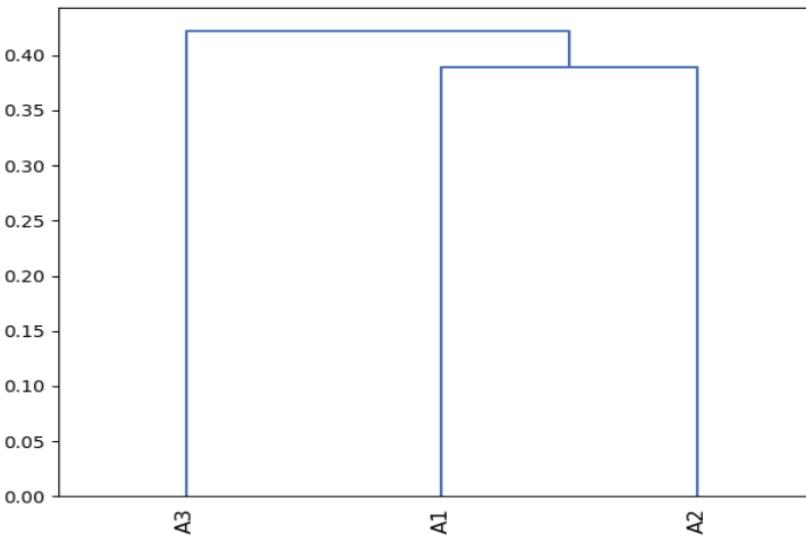
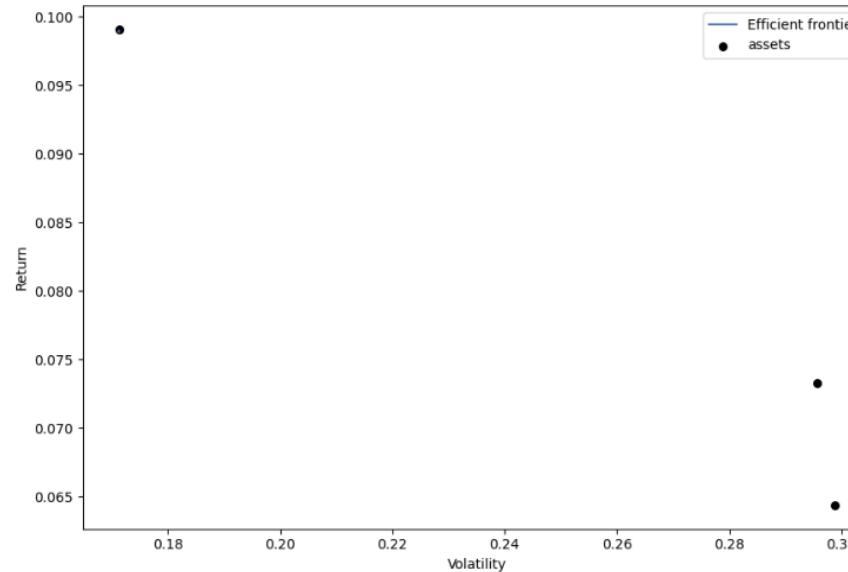


HRP portfolio weights



# Portfolio optimization

## Overall



# Portfolio optimization

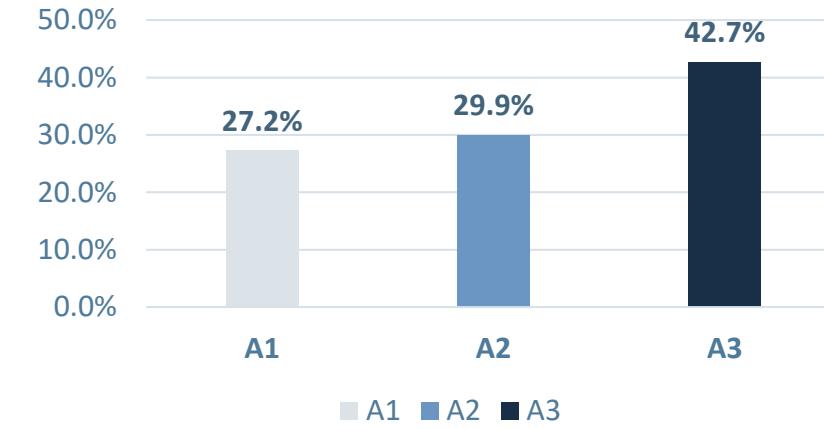
## Low Volatility

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

**Tangency portfolio weights**

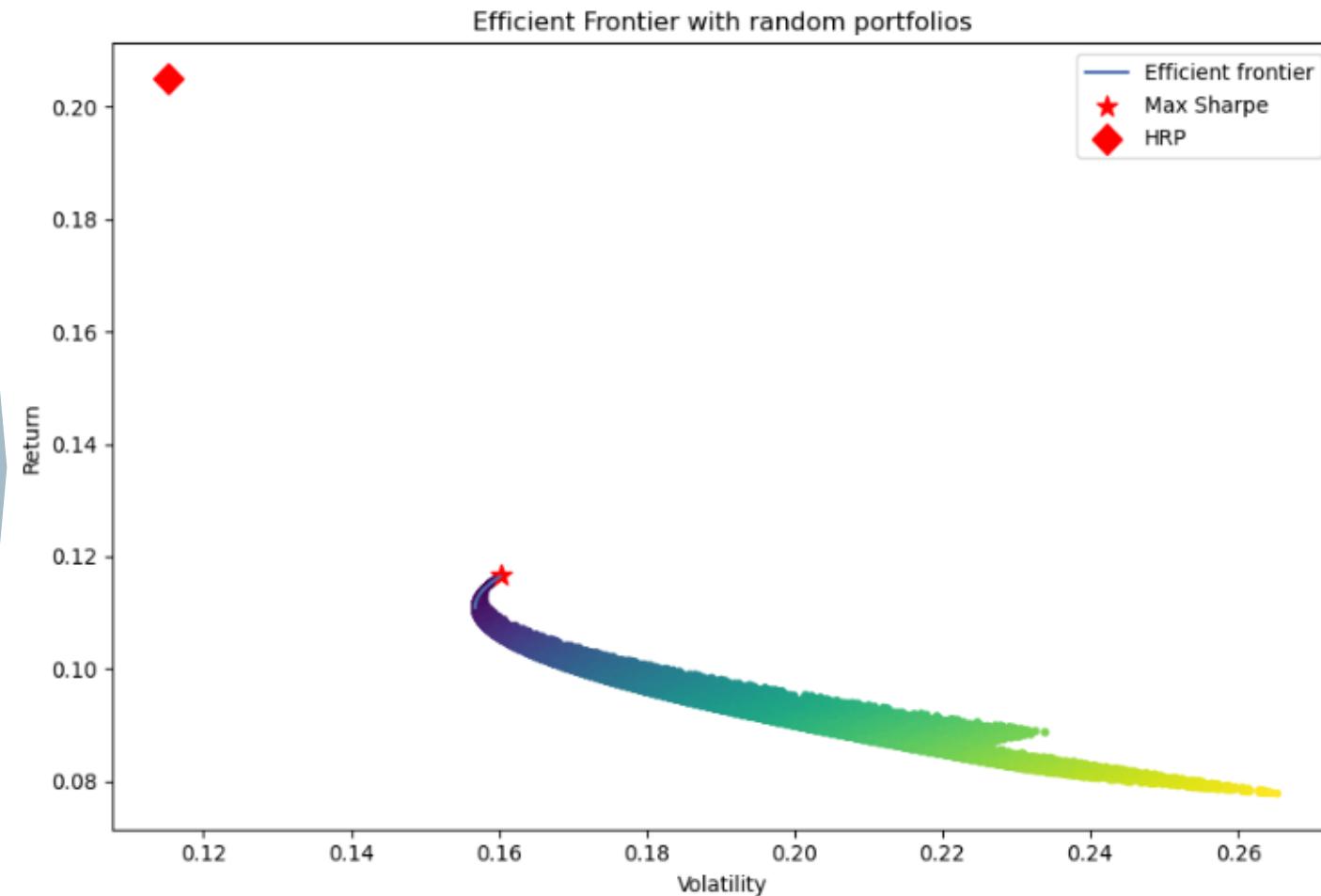
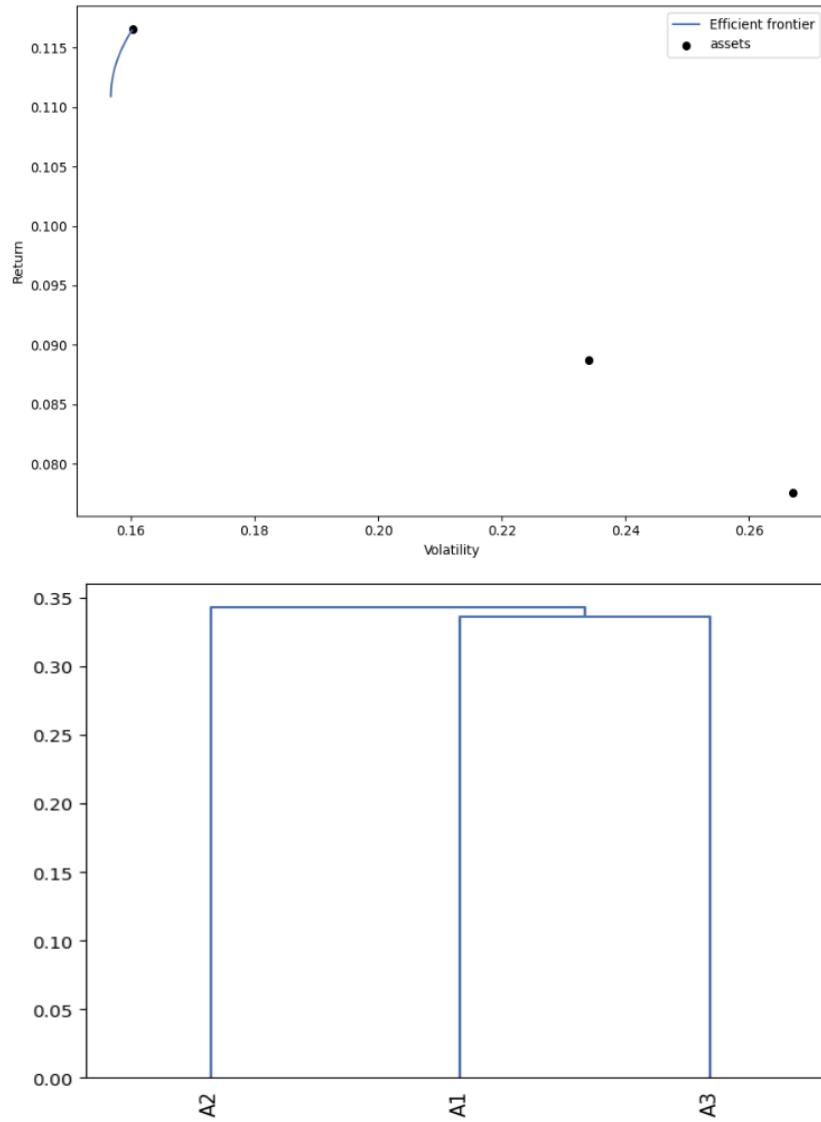


**HRP portfolio weights**



# Portfolio optimization

## Low Volatility



# Portfolio optimization

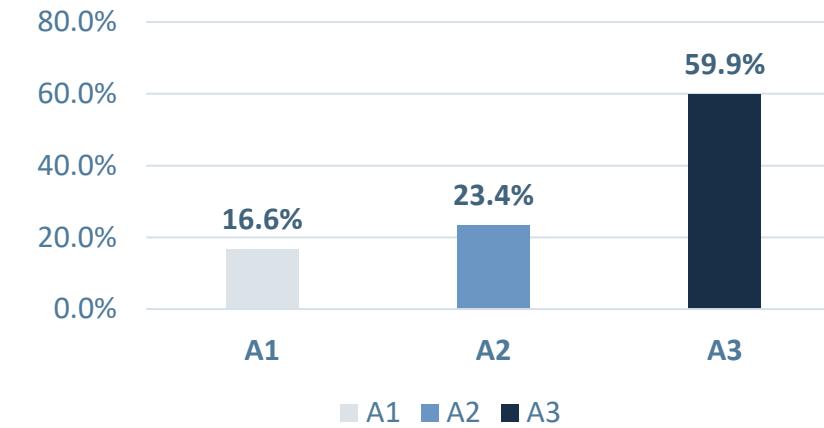
## High Volatility

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

Tangency portfolio weights

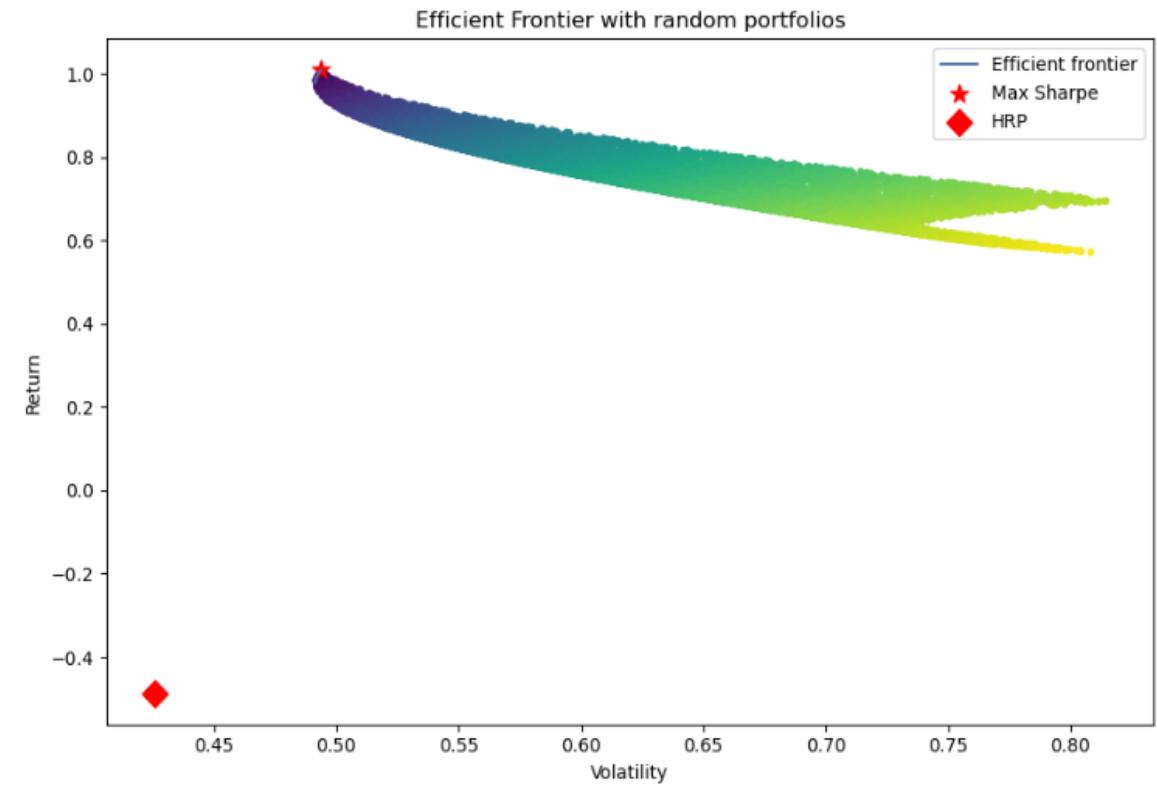
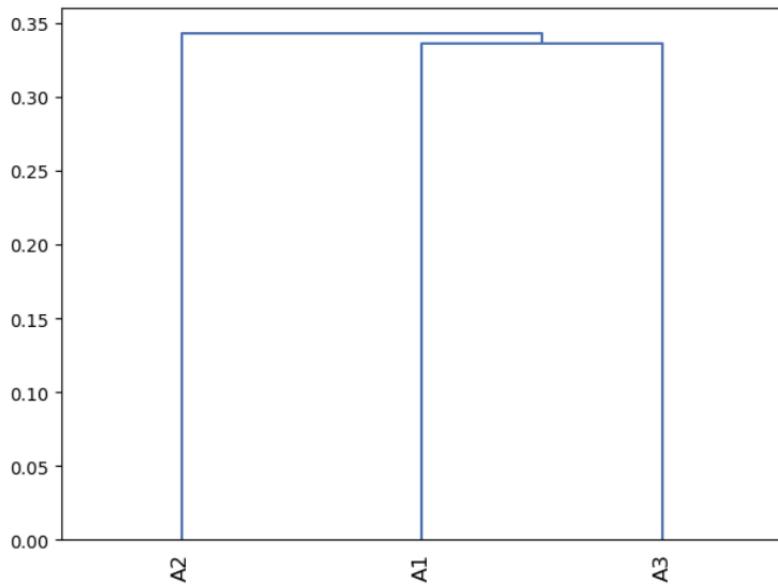
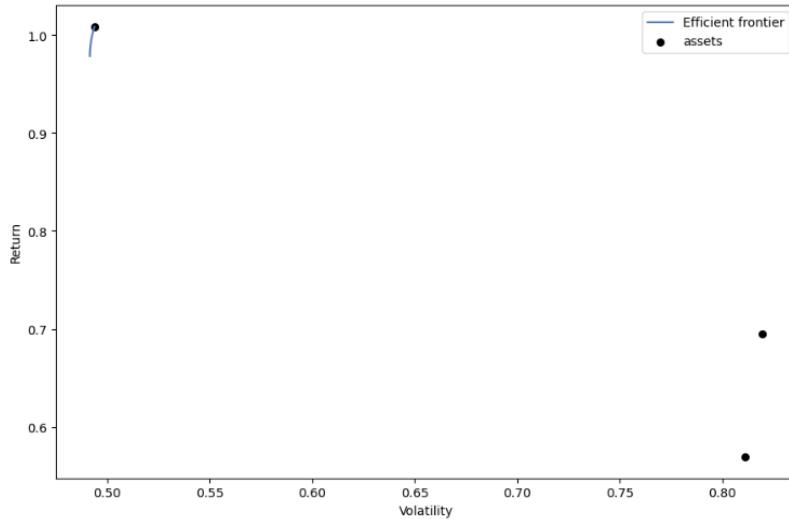


HRP portfolio weights



# Portfolio optimization

## High Volatility

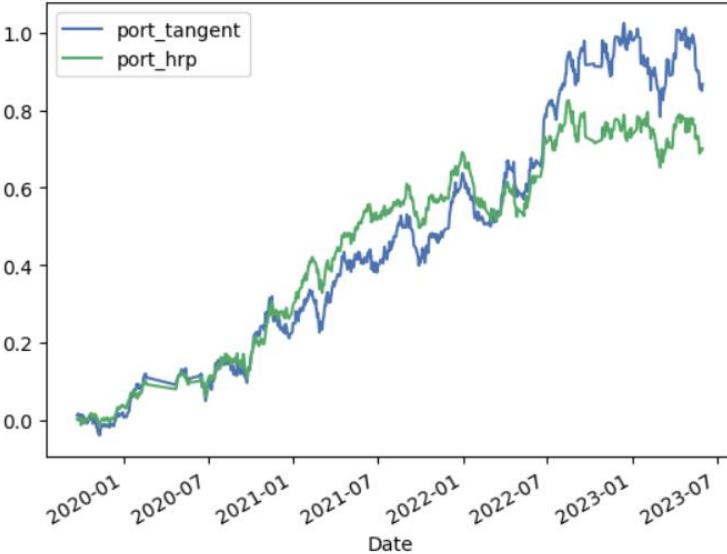


# Findings and Recommendations

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

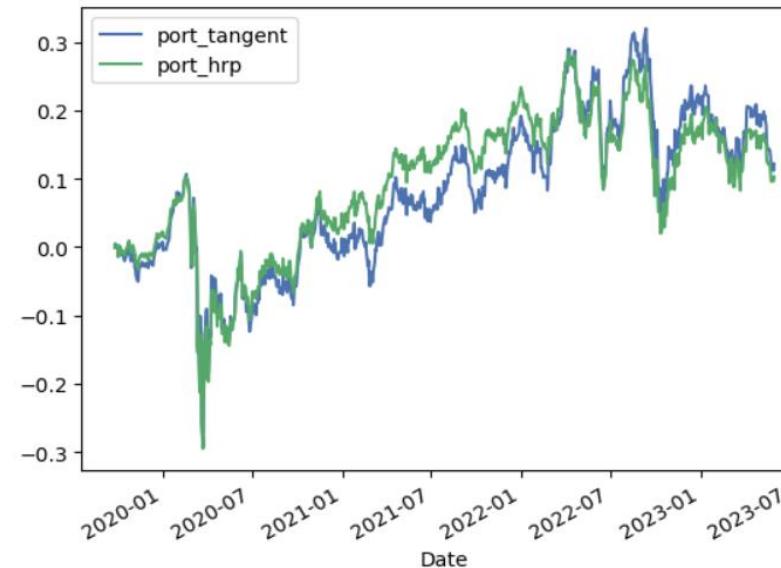
**Low-volatility State**

Tangent std: 0.317    HRP std: 0.257



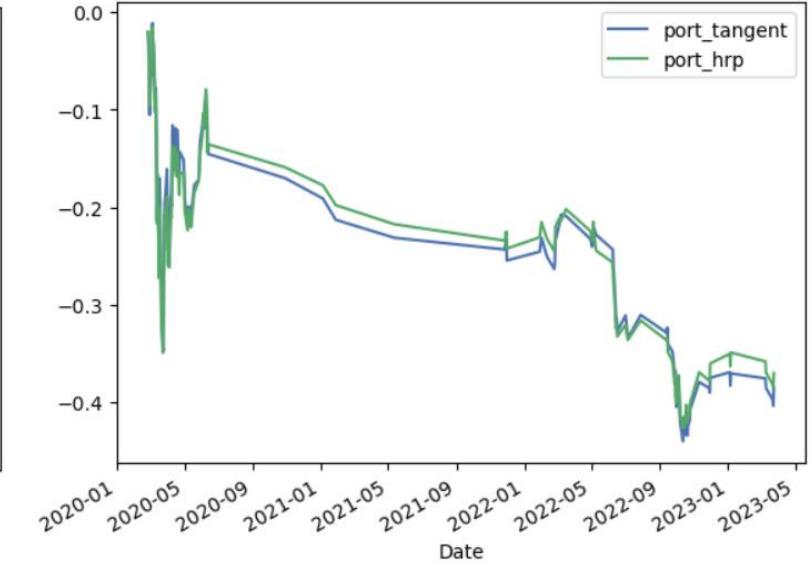
**Original state**

Tangent std: 0.193    HRP std: 0.173



**High-volatility State**

Tangent std: 0.112    HRP std: 0.108



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

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

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



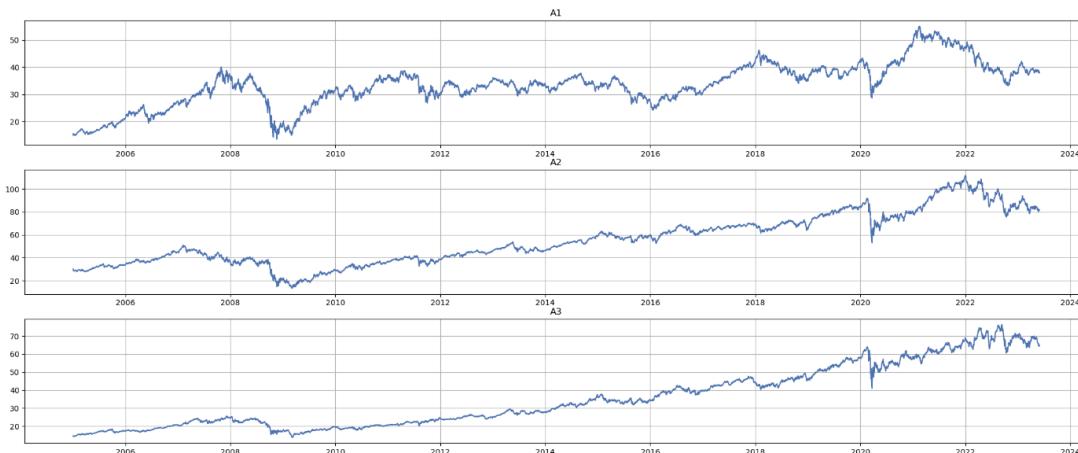
**Thank  
you!**

# Appendix

# Main part of the code (Market Behaviour)

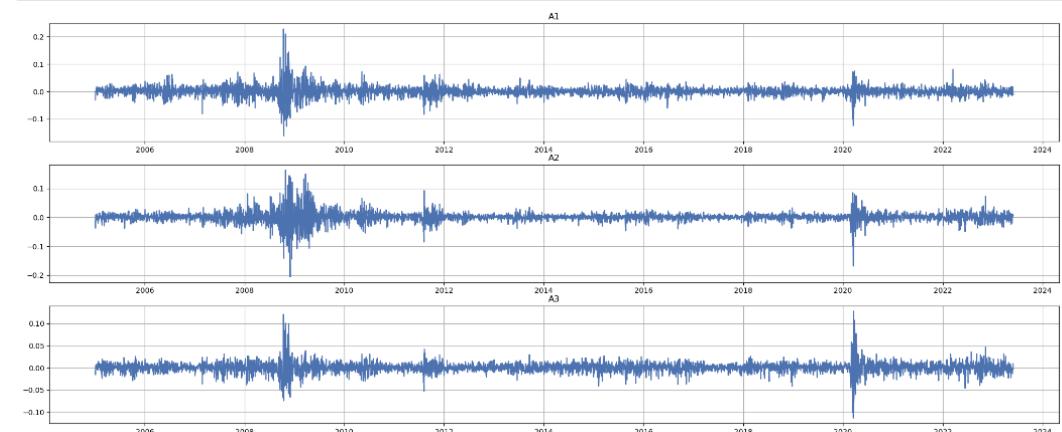
```
In [4]: tickers = list(df.columns)
```

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

```
In [15]: pd.DataFrame(data=model.means_.values, 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
```

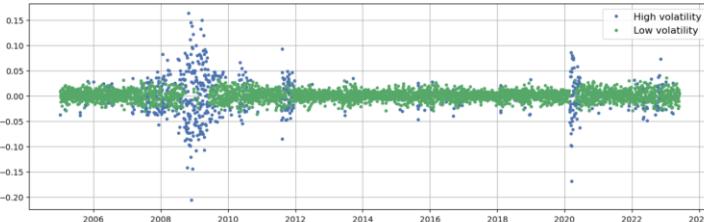
```
In [16]: # regime plotting function (will be called repeatedly for each asset)
def regime_plot(df, nasset):
    plt.figure(figsize=(15, 10))
    plt.subplot(2, 1, 1)
    for i in states:
        s = (Z == i)
        x = df.index[s]
        y = df[i].loc[nasset].iloc[s]
        plt.plot(x, y, 'o', label=states_dict[i]) # Add label based on regime
    plt.legend(fontsize=10)
    plt.grid(True)

    plt.subplot(2, 1, 2)
    for i in states:
        s = (Z == i)
        x = df.index[s]
        y = df[i].loc[nasset].pct_change().iloc[s]
        plt.plot(x, y, 'o', label=states_dict[i]) # Add label based on regime
    plt.legend(fontsize=10)
    plt.grid(True)

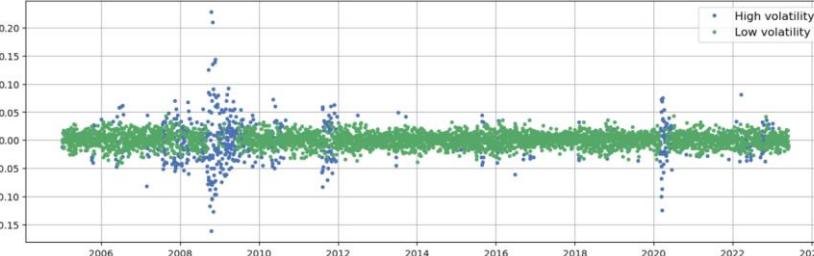
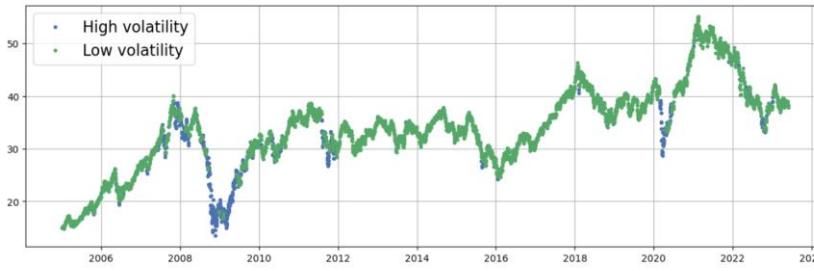
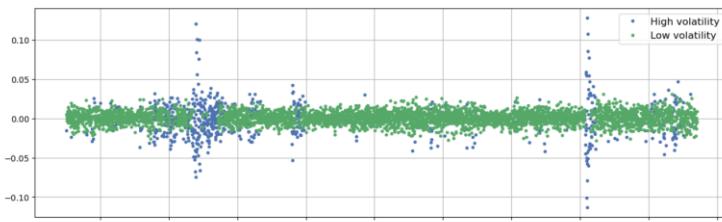
    return plt
```

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

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



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



# Main part of the code (Correlation between assets)

```
In [25]: 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)

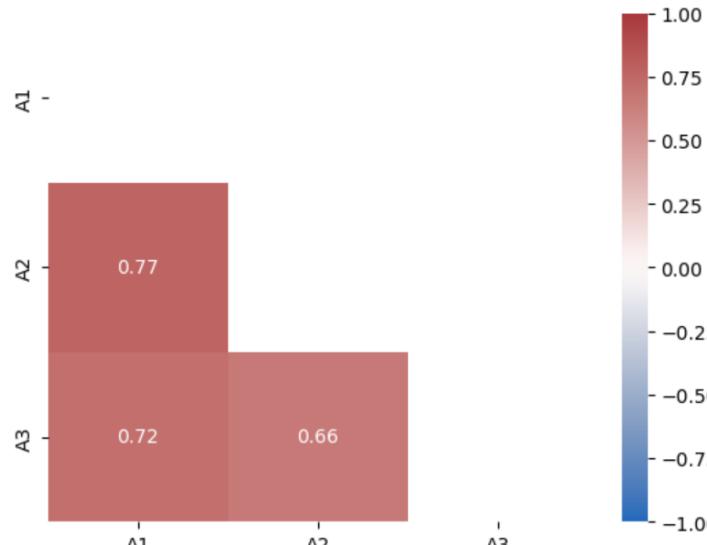
ax = avg_returns_df.pivot(columns='state').plot.barh()
plt.title('AVG Returns per Regime')

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

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

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

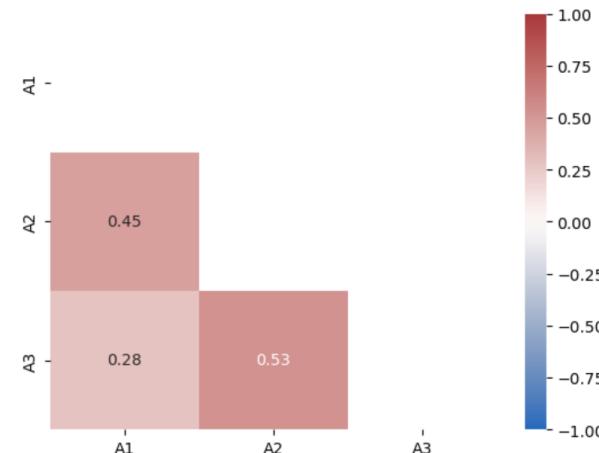
# Plot the correlation matrix as a heatmap
sns.heatmap(corr_matrix, annot=True, vmin=-1, vmax=1, cmap="vlag", mask=mask)
plt.show()
```



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

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

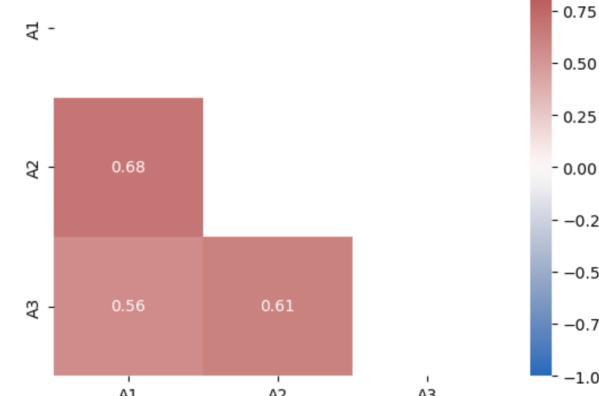
# Plot the correlation matrix as a heatmap
sns.heatmap(corr_matrix, annot=True, vmin=-1, vmax=1, cmap="vlag", mask=mask)
plt.show()
```



```
In [29]: 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_vo1_df_s = low_vo1_df.copy()
low_vo1_df_s = low_vo1_df_s[['A1', 'A2', 'A3']]
low_vo1_returns_df_s = low_vo1_returns_df.copy()
low_vo1_returns_df_s = low_vo1_returns_df_s[['A1', 'A2', 'A3']]
high_vo1_df_s = high_vo1_df.copy()
high_vo1_df_s = high_vo1_df_s[['A1', 'A2', 'A3']]
high_vo1_returns_df_s = high_vo1_returns_df.copy()
high_vo1_returns_df_s = high_vo1_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_vo1_df into train and test sets
low_vo1_train, low_vo1_test = train_test_split(low_vo1_df_s, test_size=0.2, shuffle=False)
low_vo1_returns_train, low_vo1_returns_test = train_test_split(low_vo1_returns_df_s, test_size=0.2, shuffle=False)

# Splitting high_vo1_df into train and test sets
high_vo1_train, high_vo1_test = train_test_split(high_vo1_df_s, test_size=0.2, shuffle=False)
high_vo1_returns_train, high_vo1_returns_test = train_test_split(high_vo1_returns_df_s, test_size=0.2, shuffle=False)

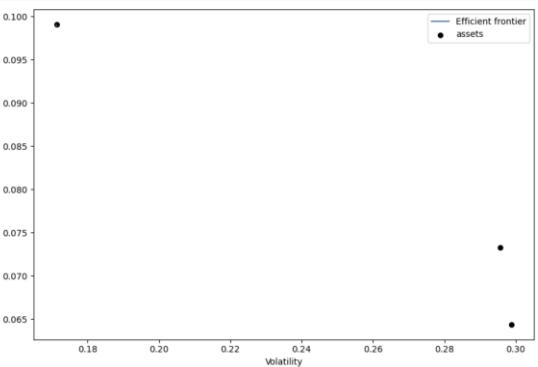
# Splitting states_df into train and test sets
states_train, states_test = train_test_split(states_df_s, test_size=0.2, shuffle=False)
states_returns_train, states_returns_test = train_test_split(states_returns_df_s, test_size=0.2, shuffle=False)

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

Out[32]:
A3    0.09494
A2    0.07264
A1    0.05498
dtype: float64

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

Out[33]:
   A1    A2    A3
A1  0.08934  0.06072  0.030934
A2  0.06072  0.09738  0.028964
A3  0.030934  0.028964  0.039349
```



```
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 = 1000
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))
sharpe = rets / stds

Out[39]:
array([0.25600859, 0.47823743, 0.30080144, ..., 0.52276217, 0.26077736,
       0.38856904])

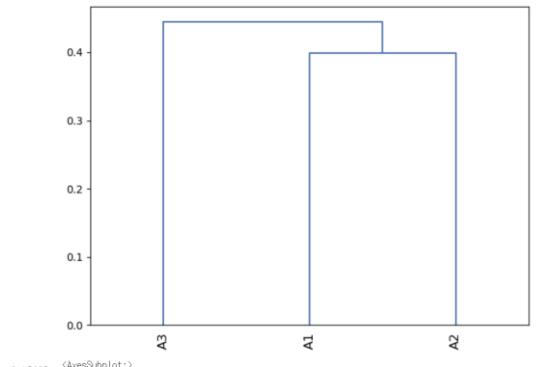
In [40]: # average Historical Risk Parity (HRP) portfolio
hrp = HRPModel(states_returns_train)
hrp.weights = hrp.optimize()
hrp.weights

Out[40]:
OrderedDict([('A1', 0.1400440170405413),
              ('A2', 0.1400442650062324),
              ('A3', 0.7169813023892261))

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

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

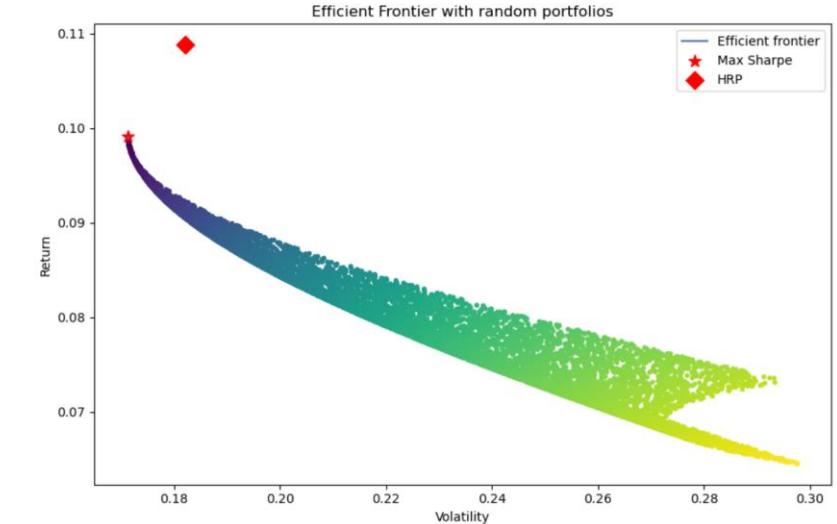
In [42]: # Hierarchical Clustering dendrogram
plotting.plot_dendrogram(hrp, showfig = True)
```



```
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=sharpe, cmap="viridis_r")
ax.scatter(std_tangent, ret_tangent, marker="x", 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)

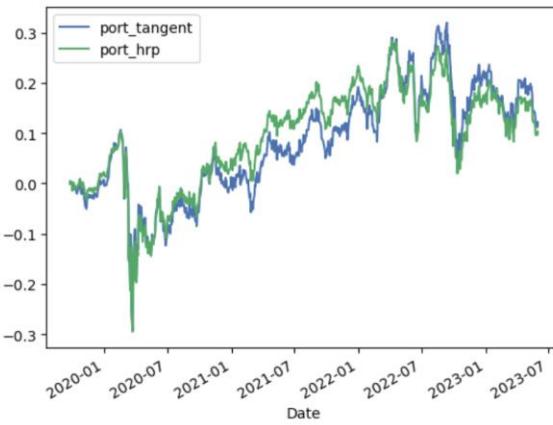
```
In [44]: # construct portfolio returns: testing time period
states_returns_test['port_tangent'] = 0
for ticker, weight in tangent_weights.items():
    states_returns_test['port_tangent'] += states_returns_test[ticker]*weight
```

```
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
```

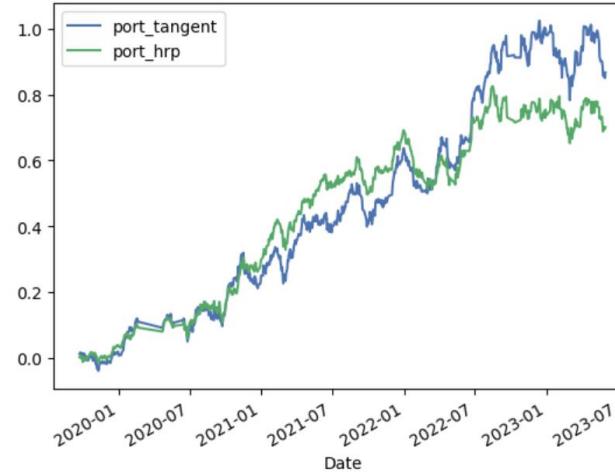
```
In [48]: # out-of-sample performance
port_equity.plot()
```

```
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'>
```

