```
!pip install PyPortfolioOpt

Collecting PyPortfolioOpt
  Downloading pyportfolioopt-1.5.6-py3-none-any.whl.metadata (22 kB)
Requirement already satisfied: cvxpy>=1.1.19 in
/usr/local/lib/python3.10/dist-packages (from PyPortfolioOpt) (1.5.4)
Requirement already satisfied: ecos<3.0.0,>=2.0.14 in
/usr/local/lib/python3.10/dist-packages (from PyPortfolioOpt) (2.0.14)
Requirement already satisfied: numpy>=1.26.0 in
/usr/local/lib/python3.10/dist-packages (from PyPortfolioOpt) (1.26.4)
Requirement already satisfied: pandas>=0.19 in
/usr/local/lib/python3.10/dist-packages (from PyPortfolioOpt) (2.2.2)
Requirement already satisfied: plotly<6.0.0,>=5.0.0 in
/usr/local/lib/python3.10/dist-packages (from PyPortfolioOpt) (5.24.1)
Requirement already satisfied: scipy>=1.3 in
/usr/local/lib/python3.10/dist-packages (from PyPortfolioOpt) (1.13.1)
Requirement already satisfied: osqp>=0.6.2 in
/usr/local/lib/python3.10/dist-packages (from cvxpy>=1.1.19-
>PyPortfolioOpt) (0.6.7.post3)
Requirement already satisfied: clarabel>=0.5.0 in
/usr/local/lib/python3.10/dist-packages (from cvxpy>=1.1.19-
>PyPortfolioOpt) (0.9.0)
Requirement already satisfied: scs>=3.2.4.post1 in
/usr/local/lib/python3.10/dist-packages (from cvxpy>=1.1.19-
>PyPortfolioOpt) (3.2.7)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.19-
>PyPortfolioOpt) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.19-
>PyPortfolioOpt) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.19-
>PyPortfolioOpt) (2024.2)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from plotly<6.0.0,>=5.0.0-
>PyPortfolioOpt) (9.0.0)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from plotly<6.0.0,>=5.0.0-
>PyPortfolioOpt) (24.2)
Requirement already satisfied: qdldl in
/usr/local/lib/python3.10/dist-packages (from osqp>=0.6.2-
>cvxpy>=1.1.19->PyPortfolioOpt) (0.1.7.post4)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2-
>pandas>=0.19->PyPortfolioOpt) (1.16.0)
Downloading pyportfolioopt-1.5.6-py3-none-any.whl (62 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.7/62.7 kB 2.7 MB/s eta
0:00:00
```

```
!conda install -c conda-forge pandoc

!pip install --upgrade pandoc

# import packages
import pandas as pd
import numpy as np
import csv
import random
from scipy import optimize
import requests
import os
from matplotlib import pyplot as plt
from matplotlib.ticker import FixedLocator, FixedFormatter
from pypfopt import HRPOpt
from pypfopt.expected_returns import mean_historical_return
from pypfopt.risk_models import CovarianceShrinkage
from pypfopt.efficient_frontier import EfficientFrontier

from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
%cd "drive/MyDrive/Final/data/"
```

```
/content/drive/MyDrive/Final/data
```

```
# select exchange - Binance
exchange = input('Please select the crypto exchange: ')
```

```
Please select the crypto exchange: Bitstamp
```

```
# select cryptocurrency - SOLUSDT (start in 2020) - MATICUSDT (died in
09 09 24) # 'SOLUSDT' 'MATICUSDT'
"""
['BTCUSD', 'ETHUSD', 'XRPUSD', 'ADAUSD', 'DOGEUSD', 'DOTUSD',
'LTCUSD']
"""
cryptos = []
crypto = "d"
while crypto!='q':
  try:
    crypto = input('Please enter the cryptocurrency code or press q to
quit: ')
    if crypto != 'q':
      cryptos.append(crypto)
  except crypto=="":
    print("Please enter the cryptocurrency code")
print(f"Your selected cryptos: {cryptos}")
```

```
Please enter the cryptocurrency code or press q to quit: BTCUSD
Please enter the cryptocurrency code or press q to quit: ETHUSD
Please enter the cryptocurrency code or press q to quit: XRPUSD
Please enter the cryptocurrency code or press q to quit: BNBUSD
Please enter the cryptocurrency code or press q to quit: ADAUSD
Please enter the cryptocurrency code or press q to quit: DOGEUSD
Please enter the cryptocurrency code or press q to quit: DOTUSD
Please enter the cryptocurrency code or press q to quit: LTCUSD
Please enter the cryptocurrency code or press q to quit: BCHUSD
Please enter the cryptocurrency code or press q to quit: q
Your selected cryptos: ['BTCUSD', 'ETHUSD', 'XRPUSD', 'BNBUSD',
'ADAUSD', 'DOGEUSD', 'DOTUSD', 'LTCUSD', 'BCHUSD']
```

```
# cryptos=['BTCUSD', 'ETHUSD', 'XRPUSD', 'ADAUSD', 'DOGEUSD',
'DOTUSD', 'LTCUSD', 'SOLUSD']
```

```
# select frequency
freq_ok = 0
while freq_ok == 0:
    freq_in = input('Please select frequency (daily/hourly): ')
    if freq_in == 'daily':
        freq = 'd'
        freq_ok = 1
    elif freq_in == 'hourly':
        freq = '1h'
        freq_ok = 1
    else:
        print("Incorrect input")
```

```
Please select frequency (daily/hourly): daily
```

Data Collection and Analysis

In this notebook, we'll use one way to scrap some cryptocurrency data from the internet. It is not the only way you can obtain data, and probabily not the best one. But it is a good application of the things seen in class.

We are using API data from cryptodatadownload.com

After creating our database, we will take a first analysis of our time series with plots and summary statistics.

Data collection

In the code below, we select the frequency, the exchange, and the code of the cryptocurrency that we select. The code in the data collection part must be run multiple times, one for every frequency, exchange and cryptocurrency code. We could download everything that we need in one run by creating a list of values for each of the three attributes and using loops.

Daily Data collection and time series analysis

```
# create paths a.k.a create relevant folders
exchange_path = os.getcwd() + '/' + exchange
if not os.path.exists(exchange_path):
    os.mkdir(exchange_path)
output_path = exchange_path + '/' + freq_in
if not os.path.exists(output_path):
    os.mkdir(output_path)

# creating empty files for each crypto
for crypto in cryptos:
    file_path = output_path + '/' + crypto + '.csv'

    try:
      with open(file_path, "w") as f:
          print(f"File created successfully at {file_path}")
    except FileExistsError:
            print(f"File already exists at {file_path}")
```

```
File created successfully at
/content/drive/MyDrive/Final/data/Bitstamp/daily/BTCUSD.csv
File created successfully at
/content/drive/MyDrive/Final/data/Bitstamp/daily/ETHUSD.csv
File created successfully at
/content/drive/MyDrive/Final/data/Bitstamp/daily/XRPUSD.csv
File created successfully at
/content/drive/MyDrive/Final/data/Bitstamp/daily/ADAUSD.csv
File created successfully at
/content/drive/MyDrive/Final/data/Bitstamp/daily/DOGEUSD.csv
File created successfully at
/content/drive/MyDrive/Final/data/Bitstamp/daily/DOTUSD.csv
File created successfully at
/content/drive/MyDrive/Final/data/Bitstamp/daily/LTCUSD.csv
File created successfully at
/content/drive/MyDrive/Final/data/Bitstamp/daily/SOLUSD.csv
```

```
# Checking the new assets
#pd.read_csv("/content/drive/MyDrive/Final/data/binance/daily/ADAUSDT.
csv").tail()

for crypto in cryptos:
    url = 'https://www.cryptodatadownload.com/cdd/' + exchange + '_' +
crypto + '_' + freq + '.csv'

    # download from API with the requests module
    response = requests.get(url)

    # save data in csv file
    if response.status_code == 200:
        # Decode the content
```

```
        decoded_content = response.content.decode('utf-8')

        # Split the content into lines
        csv_lines = decoded_content.splitlines()

        # Use the csv module to write to a file
        with open(f'{output_path}/{crypto}.csv', mode='w', newline='',
encoding='utf-8') as file:
            writer = csv.writer(file)
            for line in csv.reader(csv_lines[1:]):
                writer.writerow(line)

        print(f"CSV file saved as {crypto}.csv")
```

```
CSV file saved as BTCUSD.csv
CSV file saved as ETHUSD.csv
CSV file saved as XRPUSD.csv
CSV file saved as ADAUSD.csv
CSV file saved as DOGEUSD.csv
CSV file saved as DOTUSD.csv
CSV file saved as LTCUSD.csv
CSV file saved as SOLUSD.csv
```

Database creation

In the code below, we select the frequency and the exchange. Then we create a database with all the prices of the cryptocurrencies we downloaded with that specific frequency and exchange.

```
# load files
path = exchange_path + '/' + freq_in + '/'
file_names = [f for f in os.listdir(path) if not f.startswith('.')]

dataframes = {}
for file_name in file_names:
    name = file_name.split('/')[-1].split('.')[0]
    dataframes[name] = pd.read_csv(path+file_name)
```

We now identify the common columns for all the different cryptocurrencies to create a database for each attribute that collects values for every cryptocurrency.

```
# identifying common columns
common_columns = set.intersection(*[set(df.columns) for df in
dataframes.values()])
common_columns.remove('unix')
common_columns.remove('symbol')
common_columns.remove('date')

# create combined dataframes for each attribute
for column in common_columns:
```

```
    combined_df = pd.DataFrame()
    for name, df in dataframes.items():
        # Select relevant columns
        temp_df = df[['date', column]].copy()
        temp_df.rename(columns={column: name}, inplace=True)  # Rename
column to the dataset's name
        combined_df = combined_df.merge(temp_df, on='date',
how='outer') if not combined_df.empty else temp_df

    # Saving the combined DataFrame as a CSV file
    combined_df.to_csv(f'{exchange_path}' + f'/{freq}_{column}.csv',
index=False)

combined_df.dropna(inplace=True)
combined_df.head()
```


```
{"repr_error":"0","type":"dataframe","variable_name":"combined_df"}
```


Time series analysis

We now load only the closing prices and try to understand how each time series behaves and is distributed.

```
# load data
path = exchange_path + '/'
file_name = path + freq + '_close.csv'
daily_data = pd.read_csv(file_name)

daily_data["date"] = pd.to_datetime(daily_data["date"])
daily_data.set_index('date', inplace=True)
daily_data.sort_values(by='date', ascending=True, inplace=True)
daily_data.drop(daily_data.tail(1).index,inplace=True)

daily_data.dropna(inplace=True)

daily_data = daily_data[daily_data.index >= '2022-12-22']
daily_data = daily_data[daily_data.index < '2024-12-02']
daily_data.head()
```


```
{"summary":"{\n  \"name\": \"daily_data\",\n  \"rows\": 711,\n
\"fields\": [\n    {\n      \"column\": \"date\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2022-12-22 00:00:00\",\n        \"max\": \"2024-12-01 00:00:00\",\n
\"num_unique_values\": 711,\n        \"samples\": [\n          \"2024-
05-11 00:00:00\",\n          \"2024-01-20 00:00:00\",\n
\"2023-07-20 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"ETHUSD\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 714.6461935446534,\n        \"min\":
1189.5,\n        \"max\": 4066.5,\n        \"num_unique_values\":
```

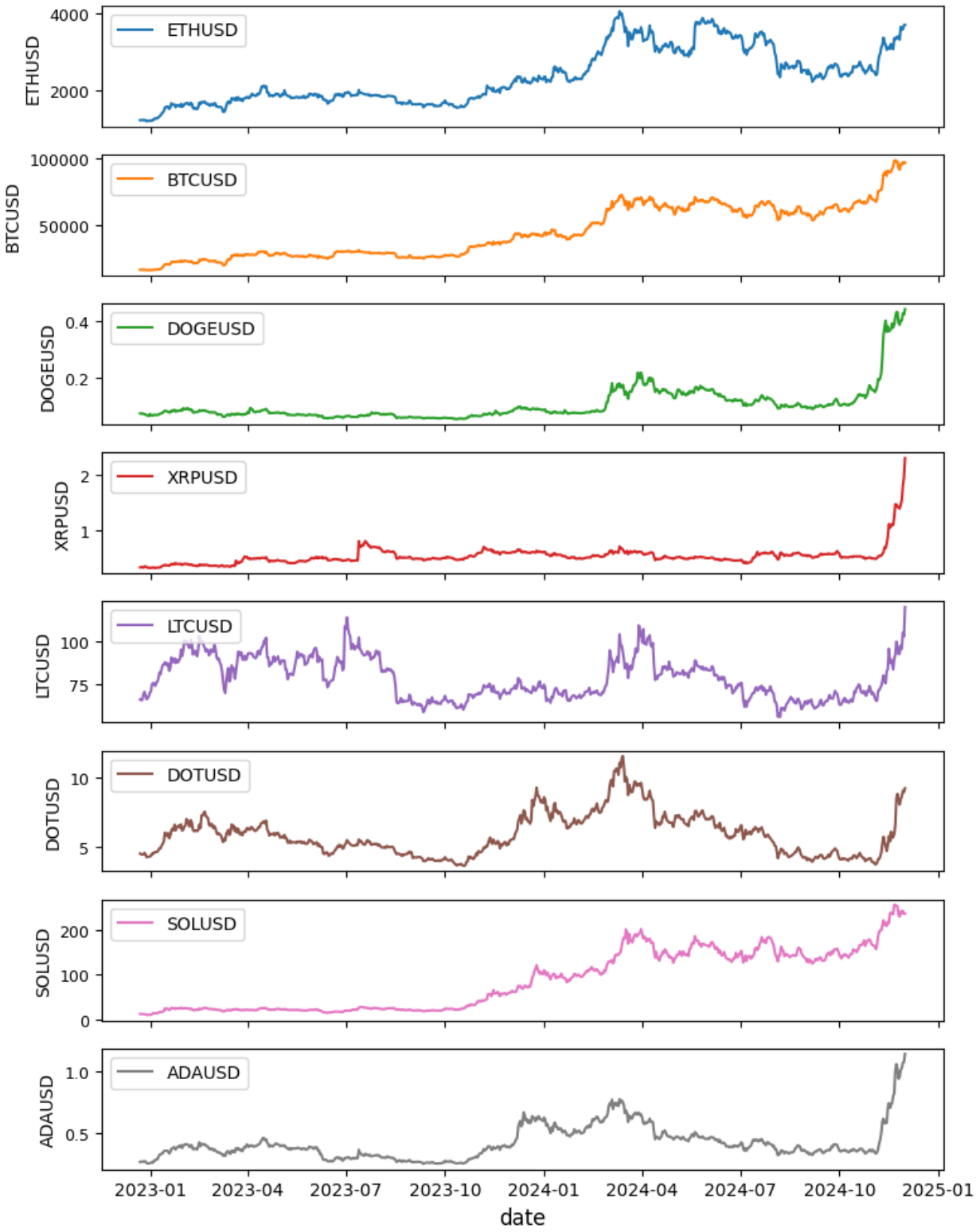
693,\n   \"samples\": [\n   2467.9,\n   2456.5,\n
1593.3\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"BTCUSD\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 19494.60043038913,\n   \"min\": 16528.0,\n
\"max\": 99013.0,\n   \"num_unique_values\": 708,\n
\"samples\": [\n   27254.0,\n   26008.0,\n
37152.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"DOGEUSD\",\n   \"properties\": {\n   \"dtype\": \"number\",\
n   \"std\": 0.061564092041057186,\n   \"min\": 0.05775,\n
\"max\": 0.44124,\n   \"num_unique_values\": 683,\n
\"samples\": [\n   0.06957,\n   0.07096,\n
0.07416\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"XRPUSD\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0.16783717976946835,\n   \"min\": 0.33789,\n
\"max\": 2.29672,\n   \"num_unique_values\": 699,\n
\"samples\": [\n   0.50651,\n   0.50191,\n
0.53042\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"LTCUSD\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 11.924132068035787,\n   \"min\": 55.99,\n
\"max\": 119.66,\n   \"num_unique_values\": 659,\n
\"samples\": [\n   70.77,\n   85.93,\n   81.18\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"DOTUSD\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
1.5198065241948144,\n   \"min\": 3.637,\n   \"max\":
11.563,\n   \"num_unique_values\": 655,\n   \"samples\": [\n
7.113,\n   5.596,\n   6.514\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"SOLUSD\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
66.76193527676652,\n   \"min\": 9.6624,\n   \"max\":
256.9647,\n   \"num_unique_values\": 711,\n   \"samples\":
[\n   145.3438,\n   92.6569,\n   25.3525\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"ADAUSD\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
0.14093932735927261,\n   \"min\": 0.24121,\n   \"max\":
1.14896,\n   \"num_unique_values\": 704,\n   \"samples\": [\
n   0.25111,\n   0.3449,\n   0.30862\n   ],\
n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe","variable_name":"daily_data"}

```
# number of assets
N_assets_tot = daily_data.shape[1]
```

Plot the data

One of the best ways to understand the behaviour of data is plotting and visualizing.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Define a color palette
colors = sns.color_palette("tab10", len(daily_data.columns))

# Plot the data
fig, axes = plt.subplots(len(daily_data.columns), 1, figsize=(8, 10),
sharex=True)

for i, column in enumerate(daily_data.columns):
    axes[i].plot(daily_data.index, daily_data[column],
color=colors[i], label=column, linewidth=1.5)
    axes[i].legend(loc="upper left", fontsize=10)
    axes[i].set_ylabel(column, fontsize=10)
    axes[i].tick_params(axis="y", labelsize=9)

axes[-1].set_xlabel("date", fontsize=12)
plt.tight_layout()
plt.show()
```



Summary statistics

Summary statistics give us a better idea of the distribution of our data. Useful statistics are:

- **Mean**. The mean of a dataset is the sum of all values divided by the total number of values. It's the most commonly used measure of central tendency and is often referred to as the "average."
- **Standard deviation**. The standard deviation is the average amount of variability in your dataset. It tells you, on average, how far each value lies from the mean. A high standard deviation means that values are generally far from the mean, while a low standard deviation indicates that values are clustered close to the mean.
- **Minimum**. The minimum value of a dataset.
- **Median**. The median is the value that's exactly in the middle of a dataset when it is ordered. It's a measure of central tendency that separates the lowest 50% from the highest 50% of values. It is also known as the 50% percentile.
- **Maximum**.The maximum value of a dataset.
- **Variance**. The variance is a measure of variability. It is calculated by taking the average of squared deviations from the mean. Variance tells you the degree of spread in your data set. The more spread the data, the larger the variance is in relation to the mean.
- **Skewness**. Skewness is a measure of the asymmetry of a distribution. A positively skewed distribution is longer on the right side of its peak, and a negatively skewed distribution is longer on the left side of its peak.
- **Kurtosis**. Kurtosis is a measure of the tailedness of a distribution, or how often outliers occur. Distributions with low kurtosis have thin tails, while distributions with high kurtosis have fat tails.
- ...

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

stats = daily_data.describe()
stats.loc['var'] = daily_data.var().tolist()
stats.loc['skew'] = daily_data.skew().tolist()
stats.loc['kurt'] = daily_data.kurtosis().tolist()

stats
```


```
{"summary":"{\n  \"name\": \"stats\",\n  \"rows\": 11,\n  \"fields\":
[\n    {\n      \"column\": \"ETHUSD\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 153509.78307577418,\n
\"min\": -0.8882929041418941,\n        \"max\": 510719.18194786226,\n
\"num_unique_values\": 11,\n        \"samples\": [\n          2222.8,\
n          711.0,\n          0.5549967703851246\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BTCUSD\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
114576778.79275997,\n        \"min\": -0.8408845278909642,\n
\"max\": 380039445.9405281,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          41560.0,\n          711.0,\n
0.47196077071940434\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DOGEUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 213.91719466040018,\n        \"min\":
```

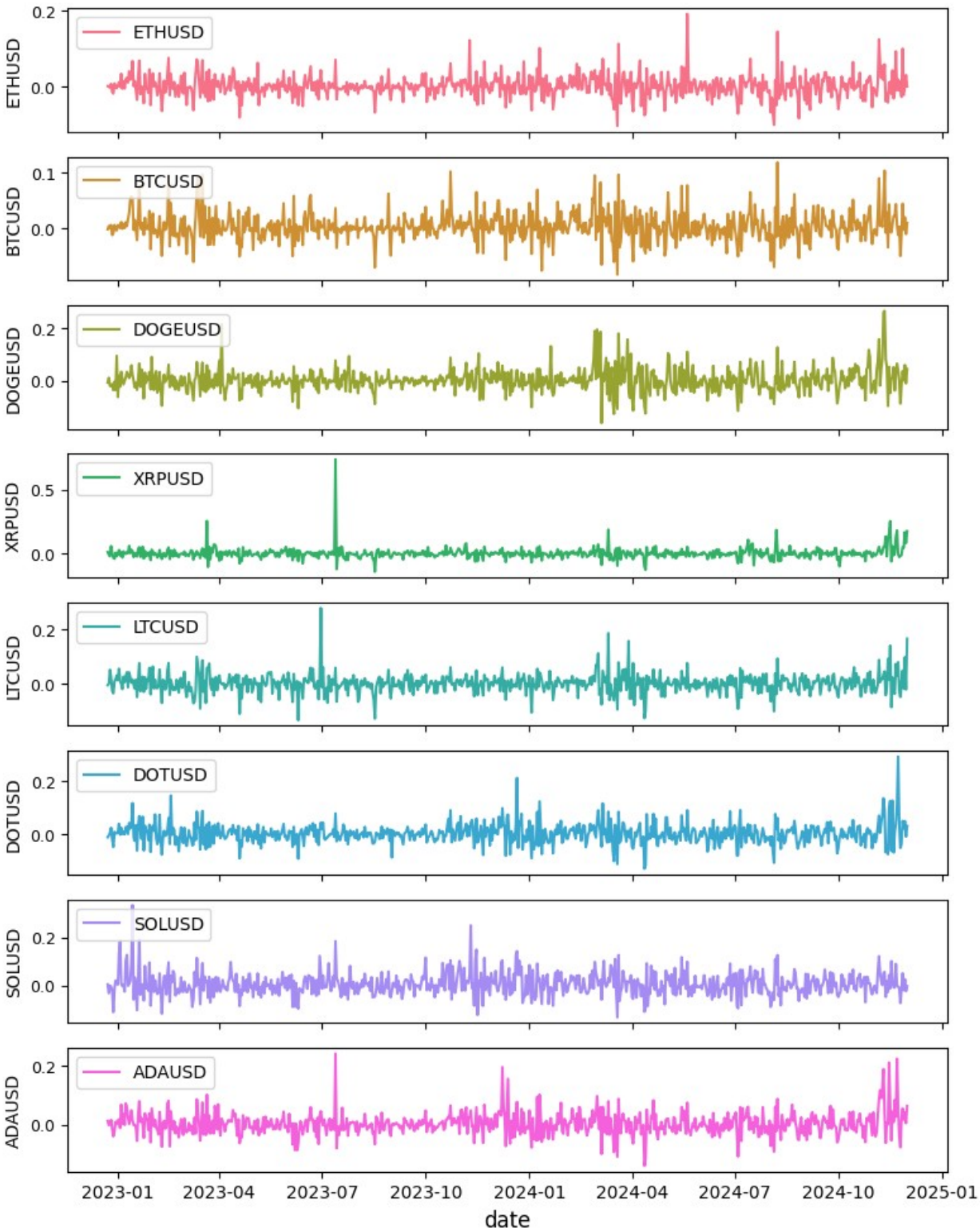
```
0.0037901374288397604,\n        \"max\": 711.0,\n
\"num_unique_values\": 11,\n        \"samples\": [\n
0.08576,\n          711.0,\n          3.1666773892948488\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"XRPUSD\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
213.26581904197462,\n        \"min\": 0.028169318912968838,\n
\"max\": 711.0,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          0.52474,\n          711.0,\n
4.866814750213273\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"LTCUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 200.43092791925534,\n        \"min\": -0.7798383293394666,\n
\"max\": 711.0,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          75.6,\n          711.0,\n
0.42822086671138004\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DOTUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 213.08622708803136,\n        \"min\": 0.6408996674818375,\n
\"max\": 711.0,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          5.667,\n          711.0,\n
0.900316256386097\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"SOLUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1318.7426224537987,\n        \"min\": -1.2113023507398142,\n
\"max\": 4457.156001899161,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          72.24,\n          711.0,\n
0.4014439902482244\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ADAUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 214.09549002036502,\n        \"min\": 0.019863893996484208,\n
\"max\": 711.0,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          0.37634,\n          711.0,\n
1.7637013823593615\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"stats"}
```


Returns

When we construct portfolios, we are more interested in the differences in prices than levels, as we want to focus on how much we can make. Moreover, thinking in differences makes all the series comparable.

```
# compute returns
daily_returns = daily_data.pct_change()[1:]

# Saving the Daily Returns
daily_returns.to_csv(f"daily_assets_returns.csv")
```

```
# Define a color palette for returns plot
return_colors = sns.color_palette("husl", len(daily_returns.columns))

# Plot the returns
fig, axes = plt.subplots(len(daily_returns.columns), 1, figsize=(8,
10), sharex=True)

for i, column in enumerate(daily_returns.columns):
    axes[i].plot(daily_returns.index, daily_returns[column],
color=return_colors[i], label=column, linewidth=1.5)
    axes[i].legend(loc="upper left", fontsize=10)
    axes[i].set_ylabel(column, fontsize=10)
    axes[i].tick_params(axis="y", labelsize=9)

axes[-1].set_xlabel("date", fontsize=12)
plt.tight_layout()
plt.show()
```




```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

```
stats_ret = daily_returns.describe()
stats_ret.loc['var'] = daily_returns.var().tolist()
stats_ret.loc['skew'] = daily_returns.skew().tolist()
stats_ret.loc['kurt'] = daily_returns.kurtosis().tolist()

stats
```


```
{"summary":"{\n  \"name\": \"stats\",\n  \"rows\": 11,\n  \"fields\":
[\n    {\n      \"column\": \"ETHUSD\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 153509.78307577418,\n
\"min\": -0.8882929041418941,\n        \"max\": 510719.18194786226,\n
\"num_unique_values\": 11,\n        \"samples\": [\n          2222.8,\
n          711.0,\n          0.5549967703851246\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BTCUSD\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
114576778.79275997,\n        \"min\": -0.8408845278909642,\n
\"max\": 380039445.9405281,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          41560.0,\n          711.0,\n
0.47196077071940434\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DOGEUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 213.91719466040018,\n        \"min\":
0.0037901374288397604,\n        \"max\": 711.0,\n
\"num_unique_values\": 11,\n        \"samples\": [\n
0.08576,\n          711.0,\n          3.1666773892948488\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"XRPUSD\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
213.26581904197462,\n        \"min\": 0.028169318912968838,\n
\"max\": 711.0,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          0.52474,\n          711.0,\n
4.866814750213273\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"LTCUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 200.43092791925534,\n        \"min\": -0.7798383293394666,\n
\"max\": 711.0,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          75.6,\n          711.0,\n
0.42822086671138004\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DOTUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 213.08622708803136,\n        \"min\": 0.6408996674818375,\n
\"max\": 711.0,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          5.667,\n          711.0,\n
0.900316256386097\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"SOLUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1318.7426224537987,\n        \"min\": -1.2113023507398142,\n
\"max\": 4457.156001899161,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          72.24,\n          711.0,\n
```

```
0.4014439902482244\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ADAUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 214.09549002036502,\n        \"min\": 0.019863893996484208,\n
\"max\": 711.0,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          0.37634,\n          711.0,\n
1.7637013823593615\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"stats"}
```


Hourly Data Collection and Analysis

Data collection

```
# select frequency
freq_ok = 0
while freq_ok == 0:
    freq_in = input('Please select frequency (daily/hourly): ')
    if freq_in == 'daily':
        freq = 'd'
        freq_ok = 1
    elif freq_in == 'hourly':
        freq = '1h'
        freq_ok = 1
    else:
        print("Incorrect input")

# create paths a.k.a create relevant folders
exchange_path = os.getcwd() + '/' + exchange
if not os.path.exists(exchange_path):
    os.mkdir(exchange_path)
output_path = exchange_path + '/' + freq_in
if not os.path.exists(output_path):
    os.mkdir(output_path)

# creating empty files for each crypto
for crypto in cryptos:
    file_path = output_path + '/' + crypto + '.csv'

    try:
      with open(file_path, "w") as f:
          print(f"File created successfully at {file_path}")
    except FileExistsError:
            print(f"File already exists at {file_path}")

for crypto in cryptos:
    url = 'https://www.cryptodatadownload.com/cdd/' + exchange + '_' +
crypto + '_' + freq + '.csv'

    # download from API with the requests module
```

```
    response = requests.get(url)

    # save data in csv file
    if response.status_code == 200:
        # Decode the content
        decoded_content = response.content.decode('utf-8')

        # Split the content into lines
        csv_lines = decoded_content.splitlines()

        # Use the csv module to write to a file
        with open(f'{output_path}/{crypto}.csv', mode='w', newline='',
encoding='utf-8') as file:
            writer = csv.writer(file)
            for line in csv.reader(csv_lines[1:]):
                writer.writerow(line)

        print(f"CSV file saved as {crypto}.csv")
```

Database creation

In the code below, we select the frequency and the exchange. Then we create a database with all the prices of the cryptocurrencies we downloaded with that specific frequency and exchange.

```
# load files
path = exchange_path + '/' + freq_in + '/'
file_names = [f for f in os.listdir(path) if not f.startswith('.')]

dataframes = {}
for file_name in file_names:
    name = file_name.split('/')[-1].split('.')[0]
    dataframes[name] = pd.read_csv(path+file_name)
```

We now identify the common columns for all the different cryptocurrencies to create a database for each attribute that collects values for every cryptocurrency.

```
# identifying common columns
common_columns = set.intersection(*[set(df.columns) for df in
dataframes.values()])
common_columns.remove('unix')
common_columns.remove('symbol')
common_columns.remove('date')

# create combined dataframes for each attribute
for column in common_columns:
    combined_df = pd.DataFrame()
    for name, df in dataframes.items():
        # Select relevant columns
        temp_df = df[['date', column]].copy()
```

```
        temp_df.rename(columns={column: name}, inplace=True)  # Rename
column to the dataset's name
        combined_df = combined_df.merge(temp_df, on='date',
how='outer') if not combined_df.empty else temp_df

    # Saving the combined DataFrame as a CSV file
    combined_df.to_csv(f'{exchange_path}' + f'/{freq}_{column}.csv',
index=False)

combined_df.head()
```

Time series analysis

We now load only the closing prices and try to understand how each time series behaves and is distributed.

```
# load data
path = exchange_path + '/'
file_name = path + freq + '_close.csv'
hourly_data = pd.read_csv(file_name)

hourly_data["date"] = pd.to_datetime(hourly_data["date"])
hourly_data.set_index('date', inplace=True)
hourly_data.sort_values(by='date', ascending=True, inplace=True)
hourly_data.drop(hourly_data.tail(1).index,inplace=True)

hourly_data = hourly_data[hourly_data.index >= '2022-12-22']
hourly_data = hourly_data[hourly_data.index < '2024-12-02']
print(hourly_data.shape)
hourly_data.head()

# number of assets
N_assets_tot = hourly_data.shape[1]
```

```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-5-2b1feea05bc9> in <cell line: 2>()
      1 # number of assets
----> 2 N_assets_tot = hourly_data.shape[1]

NameError: name 'hourly_data' is not defined
```

Plot the data

One of the best ways to understand the behaviour of data is plotting and visualizing.

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Define a color palette
colors = sns.color_palette("tab10", len(hourly_data.columns))

# Plot the data
fig, axes = plt.subplots(len(hourly_data.columns), 1, figsize=(8, 10),
sharex=True)

for i, column in enumerate(hourly_data.columns):
    axes[i].plot(hourly_data.index, hourly_data[column],
color=colors[i], label=column, linewidth=1.5)
    axes[i].legend(loc="upper left", fontsize=10)
    axes[i].set_ylabel(column, fontsize=10)
    axes[i].tick_params(axis="y", labelsize=9)

axes[-1].set_xlabel("date", fontsize=12)
plt.tight_layout()
plt.show()
```

Summary statistics

Summary statistics give us a better idea of the distribution of our data. Useful statistics are:

- **Mean**. The mean of a dataset is the sum of all values divided by the total number of values. It's the most commonly used measure of central tendency and is often referred to as the "average."
- **Standard deviation**. The standard deviation is the average amount of variability in your dataset. It tells you, on average, how far each value lies from the mean. A high standard deviation means that values are generally far from the mean, while a low standard deviation indicates that values are clustered close to the mean.
- **Minimum**. The minimum value of a dataset.
- **Median**. The median is the value that's exactly in the middle of a dataset when it is ordered. It's a measure of central tendency that separates the lowest 50% from the highest 50% of values. It is also known as the 50% percentile.
- **Maximum**.The maximum value of a dataset.
- **Variance**. The variance is a measure of variability. It is calculated by taking the average of squared deviations from the mean. Variance tells you the degree of spread in your data set. The more spread the data, the larger the variance is in relation to the mean.
- **Skewness**. Skewness is a measure of the asymmetry of a distribution. A positively skewed distribution is longer on the right side of its peak, and a negatively skewed distribution is longer on the left side of its peak.
- **Kurtosis**. Kurtosis is a measure of the tailedness of a distribution, or how often outliers occur. Distributions with low kurtosis have thin tails, while distributions with high kurtosis have fat tails.
- ...

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

```
stats = hourly_data.describe()
stats.loc['var'] = hourly_data.var().tolist()
stats.loc['skew'] = hourly_data.skew().tolist()
stats.loc['kurt'] = hourly_data.kurtosis().tolist()

stats
```

Returns

When we construct portfolios, we are more interested in the differences in prices than levels, as we want to focus on how much we can make. Moreover, thinking in differences makes all the series comparable.

```
# compute returns
hourly_returns = hourly_data.pct_change()[1:]

# Saving the Hourly Returns
hourly_returns.to_csv(f"hourly_assets_returns.csv")

# Define a color palette for returns plot
return_colors = sns.color_palette("husl", len(hourly_returns.columns))

# Plot the returns
fig, axes = plt.subplots(len(hourly_returns.columns), 1, figsize=(8,
10), sharex=True)

for i, column in enumerate(hourly_returns.columns):
    axes[i].plot(hourly_returns.index, hourly_returns[column],
color=return_colors[i], label=column, linewidth=1.5)
    axes[i].legend(loc="upper left", fontsize=10)
    axes[i].set_ylabel(column, fontsize=10)
    axes[i].tick_params(axis="y", labelsize=9)

axes[-1].set_xlabel("date", fontsize=12)
plt.tight_layout()
plt.show()

import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

stats_ret = hourly_returns.describe()
stats_ret.loc['var'] = hourly_returns.var().tolist()
stats_ret.loc['skew'] = hourly_returns.skew().tolist()
stats_ret.loc['kurt'] = hourly_returns.kurtosis().tolist()

stats
```

Dr. Martina's Portfolio Construction & Evaluation

Daily Portfolio Construction and Evaluation

In this notebook, I'll show you some of the investment strategies seen in class and how to evaluate them.

Select assets and time interval

```
daily_returns
```


```
{"summary":"{\n  \"name\": \"daily_returns\",\n  \"rows\": 710,\n
\"fields\": [\n    {\n      \"column\": \"date\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2022-12-23 00:00:00\",\n        \"max\": \"2024-12-01 00:00:00\",\n
\"num_unique_values\": 710,\n        \"samples\": [\n          \"2023-
10-13 00:00:00\",\n          \"2023-09-05 00:00:00\",\n
\"2024-11-24 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"ETHUSD\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.029310188994128333,\n        \"min\":
-0.10317302502627623,\n        \"max\": 0.19251685173727573,\n
\"num_unique_values\": 709,\n        \"samples\": [\n
0.008183943881527656,\n          0.0026385224274405594,\n
0.028558642854237037\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"BTCUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.025571886045316954,\n        \"min\": -
0.08453995533403336,\n        \"max\": 0.11903293673824722,\n
\"num_unique_values\": 710,\n        \"samples\": [\n
0.004148602182687977,\n          -0.0013943220109221555,\n
0.0023727946816671786\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DOGEUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.04462566725119533,\n        \"min\": -
0.1617751543968956,\n        \"max\": 0.26773471891462797,\n
\"num_unique_values\": 706,\n        \"samples\": [\n
0.03489240377435343,\n          -0.05885521885521883,\n
0.009252237221295312\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"XRPUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.04572051861612597,\n        \"min\": -0.13868451905595303,\
n        \"max\": 0.7324431046195652,\n        \"num_unique_values\":
710,\n        \"samples\": [\n          0.00470037685840885,\n
-0.006633097317284264,\n          -0.02493538294109643\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"LTCUSD\",\n      \"properties\":
```

{\n   \"dtype\": \"number\",\n   \"std\":
0.03591513375683094,\n   \"min\": -0.13492330086216542,\n
\"max\": 0.27948113207547176,\n   \"num_unique_values\": 709,\n
\"samples\": [\n   0.006385068762279067,\n   -
0.009260712603986865,\n   -0.04183172238947874\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"DOTUSD\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
0.038949763940624216,\n   \"min\": -0.13208677933597035,\n
\"max\": 0.2940371159111652,\n   \"num_unique_values\": 710,\n
\"samples\": [\n   0.014994547437295447,\n
0.006140765233821455,\n   0.03773363112730688\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"SOLUSD\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
0.04827527856168157,\n   \"min\": -0.13304024414574245,\n
\"max\": 0.33418109509579885,\n   \"num_unique_values\": 710,\n
\"samples\": [\n   0.025737454459983544,\n
0.04080981986290433,\n   -0.009744366483074862\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"ADAUSD\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
0.039142911893634634,\n   \"min\": -0.14117426117835852,\n
\"max\": 0.24193941080310255,\n   \"num_unique_values\": 710,\n
\"samples\": [\n   0.003142857142857114,\n
0.0057767369242780475,\n   -0.03935787486262299\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"daily_returns"}

```
# asset selection
select_asset = 1
if select_asset:
    select =
["BTCUSD","ETHUSD","XRPUSD","ADAUSD","DOGEUSD","DOTUSD","LTCUSD","SOLU
SD"] # "MATICUSDT","SOLUSDT","BNBUSDT",
"ADAUSDT","DOGEUSDT","DOTUSDT","LTCUSDT"
    daily_returns = daily_returns[select]
    daily_data = daily_data[select]
else:
    daily_returns = daily_returns
    daily_data = daily_data

# date selection
start_series_ret = daily_returns.apply(lambda series:
series.first_valid_index())
end_series_ret = daily_returns.apply(lambda series:
series.last_valid_index())
start_index_ret = max(start_series_ret)
end_index_ret = min(end_series_ret)
daily_returns = daily_returns.loc[start_index_ret:end_index_ret]
```

```
start_series_price = daily_data.apply(lambda series:
series.first_valid_index())
end_series_price = daily_data.apply(lambda series:
series.last_valid_index())
start_index_price = max(start_series_price)
end_index_price = min(end_series_price)
daily_data = daily_data.loc[start_index_price:end_index_price]

# Checking for null values in prices
daily_data.isnull().any()
```

```
BTCUSD     False
ETHUSD     False
XRPUSD     False
ADAUSD     False
DOGEUSD    False
DOTUSD     False
LTCUSD     False
SOLUSD     False
dtype: bool
```

```
# Checking for null values
daily_returns.isnull().any()
```

```
BTCUSD     False
ETHUSD     False
XRPUSD     False
ADAUSD     False
DOGEUSD    False
DOTUSD     False
LTCUSD     False
SOLUSD     False
dtype: bool
```

```
# number of assets and observations
N_obs = daily_returns.shape[0]
N_assets = daily_returns.shape[1]
```

Portfolio allocations

Portfolio style:

1. Long-short
2. Long-only

```
# select long-short
style = 1
```

Constraints for optimization:

```
# e'w = 1
e = np.ones((N_assets, 1))
Aeq = e.T
beq = 1
eq_constraint = lambda w: np.dot(Aeq, w) - beq
cons = ({'type': 'eq', 'fun': eq_constraint})

# long-short vs long-only
if style:
    lub = (0, 1)
    bnds = ((lub, ) * N_assets)
else:
    bnds = None

# initialize starting vector
w0 = np.ones((N_assets, )) / N_assets
```

Create variables for portfolio allocations: vector of expected return, covariance matrix, and correlation matrix

```
ret = np.array(daily_returns)

# Vector of expected returns
mu = np.mean(ret, axis = 0).reshape((N_assets, 1))

# Covariance matrix
Sigma = np.cov(ret.T)

# Vector of volatilities
sigmaVec = np.sqrt(np.diag(Sigma))

# Correlation matrix
corrMatrix = Sigma / np.outer(sigmaVec, sigmaVec)
corrMatrix[Sigma == 0] = 0

print(sigmaVec)
```

```
[0.02557189 0.02931019 0.04572052 0.03914291 0.04462567 0.03894976
 0.03591513 0.04827528]
```

```
# initialize dataframe to store all portfolio returns
pf_ret = pd.DataFrame()
```

Equally weighted (EW) portfolio

```
w_EW = 1 / N_assets
rets_EW = w_EW * daily_returns
ret_EW = rets_EW.dropna()
pf_ret['EW'] = ret_EW.sum(axis = 1)
```

Fundamentally weighted (FW) portfolio

```
# df = pd.read_csv(f'{os.getcwd()}/Binance/{freq}_Volume USDT.csv')
# df.columns = [col.lower() for col in df.columns]  # Convert column
names to lowercase
# df.to_csv(f'{os.getcwd()}/Binance/{freq}_Volume USDT.csv',
index=False)

freq = 'd'

# load data on volume
file_vol = path + freq + '_Volume USD.csv'
vol = pd.read_csv(file_vol)
vol["date"] = pd.to_datetime(vol["date"])
vol.set_index('date', inplace=True)
vol.sort_values(by='date', ascending=True, inplace=True)
vol.drop(vol.tail(1).index,inplace=True)

# select assets and dates
if select_asset:
    select =
["BTCUSD","ETHUSD","XRPUSD","ADAUSD","DOGEUSD","DOTUSD","LTCUSD","SOLU
SD"]  # 'bnbusdt', , 'adausdt', 'dogeusdt', 'dotusdt', 'ltcusdt'
    df_vol = vol[select]
else:
    df_vol = vol

start_series_vol = df_vol.apply(lambda series:
series.first_valid_index())
end_series_vol = df_vol.apply(lambda series:
series.last_valid_index())
start_index_vol = max(start_series_vol)
end_index_vol = min(end_series_vol)
df_vol = df_vol.loc[start_index_vol:end_index_vol]

# portfolio allocation
w_FW = np.array(df_vol.mean() / df_vol.mean().sum())
rets_FW = w_FW * daily_returns
ret_FW = rets_FW.dropna()
pf_ret['FW'] = ret_FW.sum(axis = 1)
```

Minimum variance (MV) portfolio

```
# optimization
fun_MV = lambda w: np.dot(np.dot(w.T, Sigma), w)
res_MV = optimize.minimize(fun_MV, w0, method='SLSQP', tol=1e-8,
bounds=bnds, constraints=cons)

# portfolio allocation
w_MV = res_MV.x
rets_MV = w_MV * daily_returns
```

```
ret_MV = rets_MV.dropna()
pf_ret['MV'] = ret_MV.sum(axis = 1)
```

Mean-variance portfolio

```
# risk aversion parameter
gamma = 2

# optimization
fun_meanvar = lambda w: 0.5 * gamma * np.dot(np.dot(w.T, Sigma), w) -
np.dot(w.T, mu)
res_meanvar = optimize.minimize(fun_meanvar, w0, method='SLSQP',
tol=1e-8, bounds=bnds, constraints=cons)

# portfolio allocation
w_meanvar = res_meanvar.x
rets_meanvar = w_meanvar * daily_returns
ret_meanvar = rets_meanvar.dropna()
pf_ret['meanvar'] = ret_meanvar.sum(axis = 1)


ret_EW.sum(axis = 1)
ret_FW.sum(axis = 1)
ret_MV.sum(axis = 1)
ret_meanvar.sum(axis = 1)

date
2022-12-23    0.005900
2022-12-24   -0.027310
2022-12-25   -0.004351
2022-12-26    0.008250
2022-12-27   -0.019655
                ...
2024-11-27    0.051076
2024-11-28   -0.005779
2024-11-29    0.055302
2024-11-30   -0.001689
2024-12-01    0.034000
Length: 710, dtype: float64
```

Maximum diversification (MD) portfolio

```
# optimization
fun_MD = lambda w: np.dot(-1*w.T, sigmaVec) / (np.dot(np.dot(w.T,
Sigma), w) ** 0.5)
res_MD = optimize.minimize(fun_MD, w0, method='SLSQP', tol=1e-8,
bounds=bnds, constraints=cons)

# portfolio allocation
w_MD = res_MD.x
```

```
rets_MD = w_MD * daily_returns
ret_MD = rets_MD.dropna()
pf_ret['MD'] = ret_MD.sum(axis = 1)
```

Risk parity (RP) portfolio

```
# define function to minimize
def RP_func(w, Sigma):
    x = 0
    R = np.dot(Sigma, w)
    for i in range(len(w)):
        for j in range(len(w)):
            x = x + (w[i]*R[i] - w[j]*R[j])**2
    return x

# optimization
fun_RP = lambda w: RP_func(w, Sigma)
res_RP = optimize.minimize(fun_RP, w0, method='SLSQP', tol=1e-15,
bounds=bnds, constraints=cons)

# portfolio allocation
w_RP = res_RP.x
rets_RP = w_RP * daily_returns
ret_RP = rets_RP.dropna()
pf_ret['RP'] = ret_RP.sum(axis = 1)
```

Factor investing: momentum

We create a signal based on the time-series of each asset.

```
window = 120                  # length of momentum window in days
reversalLag = -30             # Lenght of reversal window to skip in
days

# initialize
mom = np.zeros((N_obs, N_assets))
w_mom = np.ones((N_obs, N_assets)) * np.nan

# compute momentum and signal
for i in range(window, N_obs+1):
    mom[i-1, :] = np.sum(ret[i-window:i+reversalLag, :],
axis=0).reshape((1,N_assets))
    #w_mom[i-1, :] = (mom[i-1, :] - np.mean(mom[i-1, :])) /
np.std(mom[i-1, :], ddof=1)
    w_mom[i-1, :] = mom[i-1, :] / np.sum(mom[i-1, :])

ret_mom = w_mom * daily_returns
ret_mom = ret_mom.dropna()
pf_ret['mom'] = ret_mom.sum(axis = 1)
```

Portfolio evaluation

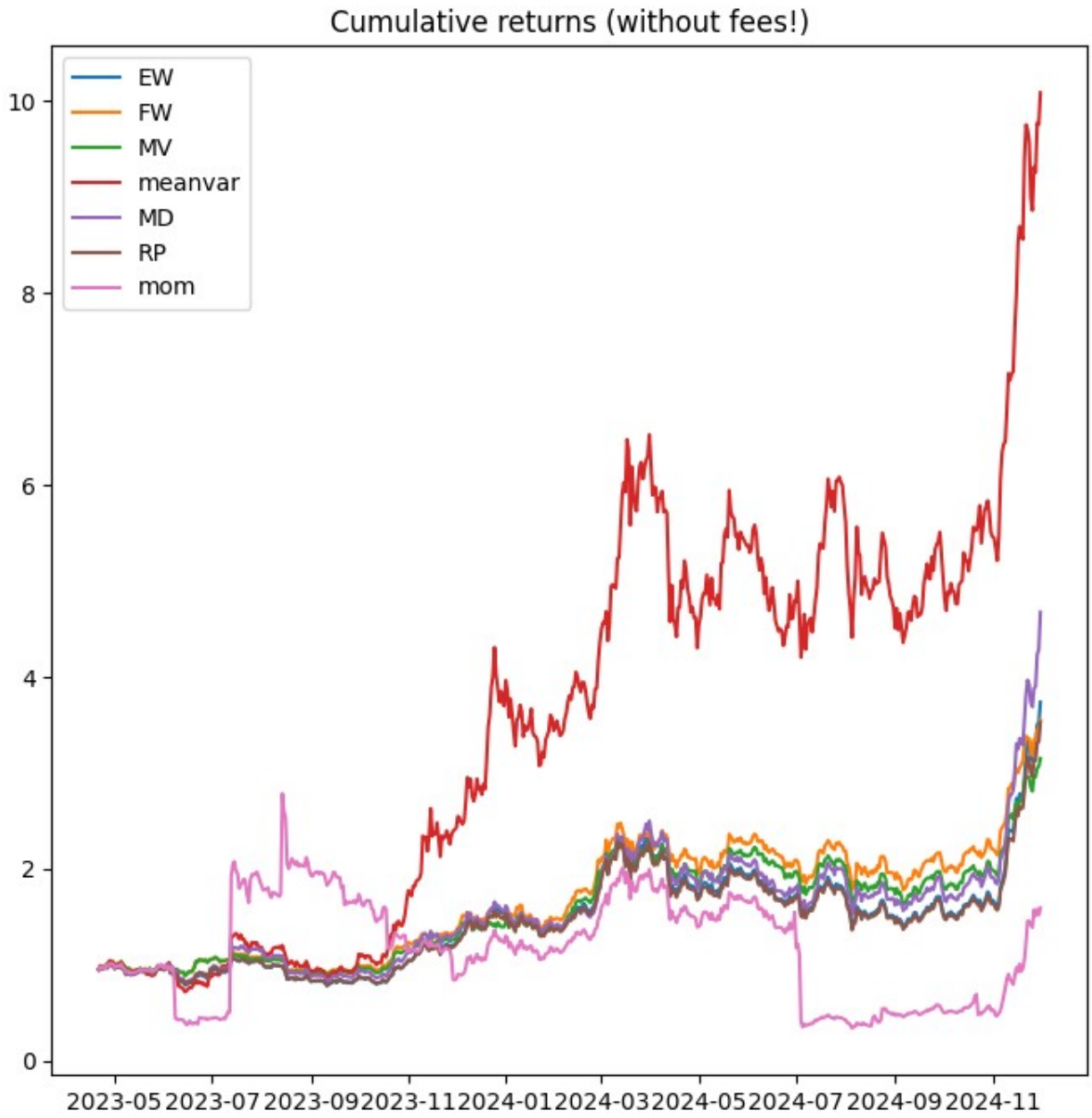
Plot cumulative returns

```
pf_ret.head()
```


```
{"summary":"{\n  \"name\": \"pf_ret\",\n  \"rows\": 710,\n
\"fields\": [\n    {\n      \"column\": \"date\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2022-12-23 00:00:00\",\n        \"max\": \"2024-12-01 00:00:00\",\n
\"num_unique_values\": 710,\n        \"samples\": [\n          \"2023-
10-13 00:00:00\",\n          \"2023-09-05 00:00:00\",\n
\"2024-11-24 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"EW\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.030387093703226897,\n        \"min\":
-0.10501091308083613,\n        \"max\": 0.18492226299221762,\n
\"num_unique_values\": 710,\n        \"samples\": [\n
0.009623727552751177,\n          0.007004500096984459,\n          -
0.008616892044159843\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"FW\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.025377094738034952,\n        \"min\": -
0.08895333031392846,\n        \"max\": 0.11353857474175466,\n
\"num_unique_values\": 710,\n        \"samples\": [\n
0.005185757965443489,\n          -0.00046669500703445705,\n          -
0.00237807183333395\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MV\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.025179315886149075,\n        \"min\": -
0.08981630752932694,\n        \"max\": 0.11681584345292113,\n
\"num_unique_values\": 710,\n        \"samples\": [\n
0.005194423928445085,\n          -0.0011866046153810304,\n          -
0.0036745108219105585\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"meanvar\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.04285111312368396,\n        \"min\": -
0.12461458627784987,\n        \"max\": 0.29819590125017603,\n
\"num_unique_values\": 710,\n        \"samples\": [\n
0.021355496734318916,\n          0.030990097082938352,\n          -
0.012908607885309516\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.03236773324466795,\n        \"min\": -0.10852147862229264,\
n        \"max\": 0.2685102574715663,\n        \"num_unique_values\":
710,\n        \"samples\": [\n          0.010792068212245499,\n
0.008095685769711168,\n          -0.012106004352253084\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"RP\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.029469230949046417,\n
```

\"min\": -0.10308634866335649,\n   \"max\": 0.17353350071866458,\n   \"num_unique_values\": 710,\n   \"samples\": [\n   0.008969136204484288,\n   0.0054148191792798145,\n   -0.008394800071237449\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"mom\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.13134528738399362,\n   \"min\": -0.6262844109688537,\n   \"max\": 2.7951881579801947,\n   \"num_unique_values\": 591,\n   \"samples\": [\n   0.031903145319604226,\n   0.13292067346563802,\n   -0.010161265749818174\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"pf_ret"}

```
pf_ret = pf_ret.dropna()


plt.figure(figsize=(8, 8))
plt.plot((1+pf_ret['EW']).cumprod(), label='EW')
plt.plot((1+pf_ret['FW']).cumprod(), label='FW')
plt.plot((1+pf_ret['MV']).cumprod(), label='MV')
plt.plot((1+pf_ret['meanvar']).cumprod(), label='meanvar')
plt.plot((1+pf_ret['MD']).cumprod(), label='MD')
plt.plot((1+pf_ret['RP']).cumprod(), label='RP')
plt.plot((1+pf_ret['mom']).cumprod(), label='mom')
plt.legend()
plt.title('Cumulative returns (without fees!)')
plt.show()
```



Annualized return and volatility, Sharpe ratio, maximum drawdown

```
# annualized return and volatility
freq = 'daily'
if freq == 'monthly':
    ann_mean = ((1+pf_ret.mean()) ** 12) - 1
    ann_vol = pf_ret.std() * (12**0.5)
elif freq == 'daily':
    ann_mean = ((1+pf_ret.mean()) ** 365) - 1
    ann_vol = pf_ret.std() * (365**0.5)

# Sharpe ratio
```

```
sharpe_ratio = ann_mean / ann_vol

# maximum drawdown
cummulative_return = (1 + pf_ret).cumprod()
previous_peaks = cummulative_return.cummax()
drawdown = (previous_peaks - cummulative_return)/previous_peaks
max_drawdown = abs(drawdown.max())
```

Summary table for comparison

```
summ_tab_martina = pd.DataFrame(columns=['EW', 'FW', 'MV', 'meanvar',
'MD', 'RP'])
summ_tab_martina.loc['Annualized return'] = ann_mean
summ_tab_martina.loc['Annualized volatility'] = ann_vol
summ_tab_martina.loc['Sharpe ratio'] = sharpe_ratio
summ_tab_martina.loc['Max DD'] = max_drawdown

summ_tab_martina
```


```
{"summary":"{\n  \"name\": \"summ_tab_martina\",\n  \"rows\": 4,\n
\"fields\": [\n    {\n      \"column\": \"EW\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.1395962648726128,\n        \"min\": 0.41140709212069104,\n
\"max\": 2.875077305525959,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.5788958537608414,\n
0.41140709212069104,\n          1.6643703314108693\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"FW\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.251398058277812,\n
\"min\": 0.2810951226875451,\n        \"max\": 3.022045649940414,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.48051308252995534,\n          0.2810951226875451,\n
1.4521324707991106\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MV\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.0793705404245364,\n        \"min\": 0.3014373592709017,\n
\"max\": 2.670030432572351,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.47690163294507265,\n
0.3014373592709017,\n          1.273341873306793\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"meanvar\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
2.7601841482010974,\n        \"min\": 0.355388788082379,\n
\"max\": 5.896203324680888,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.7814417848248129,\n
0.355388788082379,\n          4.607539649728629\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"MD\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.4208021664175616,\n
```

\"min\": 0.39132375808570913,\n   \"max\": 3.4337897726733284,\n\"num_unique_values\": 4,\n   \"samples\": [\n0.6197976523154216,\n   0.39132375808570913,\n2.128254839647634\n   ],\n   \"semantic_type\": \"\",\n\"description\": \"\"\n   }\n   },\n   {\n   \"column\":\"RP\",\n   \"properties\": {\n   \"dtype\": \"number\",\n\"std\": 1.0833625909365272,\n   \"min\": 0.4042561296208759,\n\"max\": 2.753839813581291,\n   \"num_unique_values\": 4,\n\"samples\": [\n   0.5612370606435073,\n0.4042561296208759,\n   1.545556962457428\n   ],\n\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"summ_tab_martina"}

```
pf_ret.to_csv("professor_daily_portfolio_returns.csv")

professor_daily_returns = pf_ret.copy()
professor_daily_returns
```

{"summary":"{\n \"name\": \"professor_daily_returns\",\n \"rows\": 591,\n \"fields\": [\n   {\n   \"column\": \"date\",\n\"properties\": {\n   \"dtype\": \"date\",\n   \"min\":\"2023-04-21 00:00:00\",\n   \"max\": \"2024-12-01 00:00:00\",\n\"num_unique_values\": 591,\n   \"samples\": [\n   \"2024-09-07 00:00:00\",\n   \"2024-11-27 00:00:00\",\n\"2023-11-02 00:00:00\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"EW\",\n   \"properties\": {\n   \"dtype\":\"number\",\n   \"std\": 0.030300793854671035,\n   \"min\":-0.10501091308083613,\n   \"max\": 0.18492226299221762,\n\"num_unique_values\": 591,\n   \"samples\": [\n0.01716262771057124,\n   0.054871833314135976,\n   -0.006971708403715318\n   ],\n   \"semantic_type\": \"\",\n\"description\": \"\"\n   }\n   },\n   {\n   \"column\":\"FW\",\n   \"properties\": {\n   \"dtype\": \"number\",\n\"std\": 0.02515120425137443,\n   \"min\": -0.08895333031392846,\n   \"max\": 0.11353857474175466,\n   \"num_unique_values\":591,\n   \"samples\": [\n   0.006786829638976232,\n0.05231933865718494,\n   -0.0146166364834659\n   ],\n\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"MV\",\n   \"properties\": {\n\"dtype\": \"number\",\n   \"std\": 0.024962172340578823,\n\"min\": -0.08981630752932694,\n   \"max\": 0.11681584345292113,\n   \"num_unique_values\": 591,\n   \"samples\": [\n0.007351084312805385,\n   0.05773559608713896,\n   -0.015747777160901956\n   ],\n   \"semantic_type\": \"\",\n\"description\": \"\"\n   }\n   },\n   {\n   \"column\":\"meanvar\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.04090253242889013,\n   \"min\": -0.12461458627784987,\n   \"max\": 0.29819590125017603,\n\"num_unique_values\": 591,\n   \"samples\": [\n

```
0.01788938037298414,\n          0.05107592187089015,\n          -
0.018328910343751996\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.03244169183871434,\n        \"min\": -0.10852147862229264,\
n        \"max\": 0.2685102574715663,\n        \"num_unique_values\":
591,\n        \"samples\": [\n          0.011962253682505936,\n
0.05141474490971207,\n          -0.013045464926076297\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"RP\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.029376490378502627,\n
\"min\": -0.10308634866335649,\n        \"max\": 0.17353350071866458,\
n        \"num_unique_values\": 591,\n        \"samples\": [\n
0.015665612763466587,\n          0.05593953263326537,\n          -
0.008046329228774761\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"mom\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.13134528738399362,\n        \"min\": -0.6262844109688537,\n
\"max\": 2.7951881579801947,\n        \"num_unique_values\": 591,\n
\"samples\": [\n          0.031903145319604226,\n
0.13292067346563802,\n          -0.010161265749818174\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\
n}","type":"dataframe","variable_name":"professor_daily_returns"}
```


```
professor_daily_returns.isnull().any()
```

```
EW          False
FW          False
MV          False
meanvar     False
MD          False
RP          False
mom         False
dtype: bool
```

Hourly Portfolio Construction and Evaluation

In this notebook, I'll show you some of the investment strategies seen in class and how to evaluate them.

Select assets and time interval

```
# Define a color palette for returns plot
return_colors = sns.color_palette("husl", len(hourly_data.columns))

# Plot the returns
fig, axes = plt.subplots(len(hourly_data.columns), 1, figsize=(8, 10),
sharex=True)
```

```
for i, column in enumerate(hourly_data.columns):
    axes[i].plot(hourly_data.index, hourly_data[column],
color=return_colors[i], label=column, linewidth=1.5)
    axes[i].legend(loc="upper left", fontsize=10)
    axes[i].set_ylabel(column, fontsize=10)
    axes[i].tick_params(axis="y", labelsize=9)

axes[-1].set_xlabel("date", fontsize=12)
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-157-0e5ecd19c43e> in <cell line: 2>()
      1 # Define a color palette for returns plot
----> 2 return_colors = sns.color_palette("husl",
len(hourly_data.columns))
      3
      4 # Plot the returns
      5 fig, axes = plt.subplots(len(hourly_data.columns), 1,
figsize=(8, 10), sharex=True)

NameError: name 'hourly_data' is not defined
```

```
# Define a color palette for returns plot
return_colors = sns.color_palette("husl", len(hourly_returns.columns))

# Plot the returns
fig, axes = plt.subplots(len(hourly_returns.columns), 1, figsize=(8,
10), sharex=True)

for i, column in enumerate(hourly_returns.columns):
    axes[i].plot(hourly_returns.index, hourly_returns[column],
color=return_colors[i], label=column, linewidth=1.5)
    axes[i].legend(loc="upper left", fontsize=10)
    axes[i].set_ylabel(column, fontsize=10)
    axes[i].tick_params(axis="y", labelsize=9)

axes[-1].set_xlabel("date", fontsize=12)
plt.tight_layout()
plt.show()

# number of assets and observations
N_obs = hourly_returns.shape[0]
N_assets = hourly_returns.shape[1]
```

Portfolio allocations

Portfolio style:

1. Long-short
2. Long-only

```
# select long-short
style = 1
```

Constraints for optimization:

```
# e'w = 1
e = np.ones((N_assets, 1))
Aeq = e.T
beq = 1
eq_constraint = lambda w: np.dot(Aeq, w) - beq
cons = ({'type': 'eq', 'fun': eq_constraint})

# long-short vs long-only
if style:
    lub = (0, 1)
    bnds = ((lub, ) * N_assets)
else:
    bnds = None

# initialize starting vector
w0 = np.ones((N_assets, )) / N_assets
```

Create variables for portfolio allocations: vector of expected return, covariance matrix, and correlation matrix

```
ret = np.array(hourly_returns)

# Vector of expected returns
mu = np.mean(ret, axis = 0).reshape((N_assets, 1))

# Covariance matrix
Sigma = np.cov(ret.T)

# Vector of volatilities
sigmaVec = np.sqrt(np.diag(Sigma))

# Correlation matrix
corrMatrix = Sigma / np.outer(sigmaVec, sigmaVec)
corrMatrix[Sigma == 0] = 0

print(sigmaVec)

# initialize dataframe to store all portfolio returns
pf_ret = pd.DataFrame()
```

Equally weighted (EW) portfolio

```
w_EW = 1 / N_assets
ret_EW = w_EW * hourly_returns
ret_EW = ret_EW.dropna()
pf_ret['EW'] = ret_EW.sum(axis = 1)
```

Fundamentally weighted (FW) portfolio

```
freq='1h'

# load data on volume
file_vol = path + freq + '_Volume USD.csv'
vol = pd.read_csv(file_vol)
vol["date"] = pd.to_datetime(vol["date"])
vol.set_index('date', inplace=True)
vol.sort_values(by='date', ascending=True, inplace=True)
vol.drop(vol.tail(1).index,inplace=True)

# select assets and dates
if select_asset:
    select = ["BTCUSD","ETHUSD","XRPUSD"]
    df_vol = vol[select]
else:
    df_vol = vol

start_series_vol = df_vol.apply(lambda series:
series.first_valid_index())
end_series_vol = df_vol.apply(lambda series:
series.last_valid_index())
start_index_vol = max(start_series_vol)
end_index_vol = min(end_series_vol)
df_vol = df_vol.loc[start_index_vol:end_index_vol]

# portfolio allocation
w_FW = np.array(df_vol.mean() / df_vol.mean().sum())
ret_FW = w_FW * hourly_returns
ret_FW = ret_FW.dropna()
pf_ret['FW'] = ret_FW.sum(axis = 1)
```

Minimum variance (MV) portfolio

```
# optimization
fun_MV = lambda w: np.dot(np.dot(w.T, Sigma), w)
res_MV = optimize.minimize(fun_MV, w0, method='SLSQP', tol=1e-8,
bounds=bnds, constraints=cons)

# portfolio allocation
w_MV = res_MV.x
ret_MV = w_MV * hourly_returns
```

```
ret_MV = ret_MV.dropna()
pf_ret['MV'] = ret_MV.sum(axis = 1)
```

Mean-variance portfolio

```
# risk aversion parameter
gamma = 2

# optimization
fun_meanvar = lambda w: 0.5 * gamma * np.dot(np.dot(w.T, Sigma), w) -
np.dot(w.T, mu)
res_meanvar = optimize.minimize(fun_meanvar, w0, method='SLSQP',
tol=1e-8, bounds=bnds, constraints=cons)

# portfolio allocation
w_meanvar = res_meanvar.x
ret_meanvar = w_meanvar * hourly_returns
ret_meanvar = ret_meanvar.dropna()
pf_ret['meanvar'] = ret_meanvar.sum(axis = 1)
```

Maximum diversification (MD) portfolio

```
# optimization
fun_MD = lambda w: np.dot(-1*w.T, sigmaVec) / (np.dot(np.dot(w.T,
Sigma), w) ** 0.5)
res_MD = optimize.minimize(fun_MD, w0, method='SLSQP', tol=1e-8,
bounds=bnds, constraints=cons)

# portfolio allocation
w_MD = res_MD.x
ret_MD = w_MD * hourly_returns
ret_MD = ret_MD.dropna()
pf_ret['MD'] = ret_MD.sum(axis = 1)
```

Risk parity (RP) portfolio

```
# define function to minimize
def RP_func(w, Sigma):
    x = 0
    R = np.dot(Sigma, w)
    for i in range(len(w)):
        for j in range(len(w)):
            x = x + (w[i]*R[i] - w[j]*R[j])**2
    return x

# optimization
fun_RP = lambda w: RP_func(w, Sigma)
res_RP = optimize.minimize(fun_RP, w0, method='SLSQP', tol=1e-15,
bounds=bnds, constraints=cons)
```

```
# portfolio allocation
w_RP = res_RP.x
ret_RP = w_RP * hourly_returns
ret_RP = ret_RP.dropna()
pf_ret['RP'] = ret_RP.sum(axis = 1)

pf_ret.head()
```

Factor investing: momentum

We create a signal based on the time-series of each asset.

```
# window = 120                  # length of momentum window in days
# reversalLag = -30             # Lenght of reversal window to skip in
days

# # initialize
# mom = np.zeros((N_obs, N_assets))
# w_mom = np.ones((N_obs, N_assets)) * np.nan

# # compute momentum and signal
# for i in range(window, N_obs+1):
#     mom[i-1, :] = np.sum(ret[i-window:i+reversalLag, :],
axis=0).reshape((1,N_assets))
#     #w_mom[i-1, :] = (mom[i-1, :] - np.mean(mom[i-1, :])) /
np.std(mom[i-1, :], ddof=1)
#     w_mom[i-1, :] = mom[i-1, :] / np.sum(mom[i-1, :])

# ret_mom = w_mom * hourly_returns
# ret_mom = ret_mom.dropna()
# pf_ret['mom'] = ret_mom.sum(axis = 1)

# import numpy as np
# import pandas as pd

# # Assuming 'hourly_returns' is your DataFrame of hourly returns with
N_obs and N_assets

# # Set new momentum window for hourly data (adjust for your needs)
# window = 120 * 24  # 120 days * 6 hours/day = 720 hours for the
momentum window
# reversalLag = -30 * 24  # 30 days * 6 hours/day = 180 hours for the
reversal window

# # Initialize arrays
# N_obs = len(hourly_returns)  # Number of hourly observations
# N_assets = hourly_returns.shape[1]  # Number of assets
# mom = np.zeros((N_obs, N_assets))  # To store momentum signals
# w_mom = np.ones((N_obs, N_assets)) * np.nan  # To store normalized
momentum weights
```

```
# # Compute momentum and signal
# for i in range(window, N_obs + 1):
#     # Momentum calculation: sum of returns over the momentum window
and reversal lag
#     mom[i-1, :] = np.sum(hourly_returns[i - window:i +
reversalLag, :], axis=0).reshape((1, N_assets))

#     # Normalize momentum (optional)
#     # w_mom[i-1, :] = (mom[i-1, :] - np.mean(mom[i-1, :])) /
np.std(mom[i-1, :], ddof=1)

#     # Weights based on momentum (sum of momentum)
#     w_mom[i-1, :] = mom[i-1, :] / np.sum(mom[i-1, :])

# # Calculate portfolio returns using momentum weights
# ret_mom = w_mom * hourly_returns.values  # Element-wise
multiplication with hourly returns
# ret_mom = pd.DataFrame(ret_mom).dropna()  # Drop any NaN values

# # Calculate total momentum strategy returns
# pf_ret = pd.DataFrame()
# pf_ret['mom'] = ret_mom.sum(axis=1)  # Sum across assets for
portfolio return

# # Display results
# print("Momentum Strategy Portfolio Returns (Hourly):")
# print(pf_ret['mom'])
```

Portfolio evaluation

Annualized return and volatility, Sharpe ratio, maximum drawdown

```
# annualized return and volatility
freq = 'hourly'
if freq == 'daily':
    ann_mean = ((1+pf_ret.mean()) ** 365) - 1
    ann_vol = pf_ret.std() * (365**0.5)
elif freq == 'hourly':
    ann_mean = ((1+pf_ret.mean()) ** (365*24)) - 1
    ann_vol = pf_ret.std() * ((365*24)**0.5)

# Sharpe ratio
sharpe_ratio = ann_mean / ann_vol

# maximum drawdown
cummulative_return = (1 + pf_ret).cumprod()
previous_peaks = cummulative_return.cummax()
drawdown = (previous_peaks - cummulative_return)/previous_peaks
max_drawdown = abs(drawdown.max())
```

Summary table for comparison

```
summ_tab = pd.DataFrame(columns=['EW', 'FW', 'MV', 'meanvar', 'MD',
'RP']) # 'mom'
summ_tab.loc['Annualized return'] = ann_mean
summ_tab.loc['Annualized volatility'] = ann_vol
summ_tab.loc['Sharpe ratio'] = sharpe_ratio
summ_tab.loc['Max DD'] = max_drawdown

summ_tab

pf_ret.dropna(inplace=True)

pf_ret.head()

pf_ret.to_csv("professor_hourly_portfolio_returns.csv")

professor_hourly_portfolio_returns = pf_ret.copy()

professor_hourly_portfolio_returns.isnull().any()

pf_ret = pf_ret.dropna()


plt.figure(figsize=(8, 8))
plt.plot((1+pf_ret['EW']).cumprod(), label='EW')
plt.plot((1+pf_ret['FW']).cumprod(), label='FW')
plt.plot((1+pf_ret['MV']).cumprod(), label='MV')
plt.plot((1+pf_ret['meanvar']).cumprod(), label='meanvar')
plt.plot((1+pf_ret['MD']).cumprod(), label='MD')
plt.plot((1+pf_ret['RP']).cumprod(), label='RP')
#plt.plot((1+pf_ret['mom']).cumprod(), label='mom')
plt.legend()
plt.title('Cumulative returns (without fees!)')
plt.show()
```

Portfolio construction and evaluation

In this notebook, I'll show you some of the investment strategies seen in class and how to evaluate them.

Select assets and time interval

```
# Group the data by month and calculate the standard deviation for
each group
hrdaily_returns_std =
professor_hourly_portfolio_returns.groupby(pd.Grouper(freq='D')).std()
* np.sqrt(24)

hrdaily_returns_std
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-364-329611bd5946> in <cell line: 1>()
----> 1 hrdaily_returns_std

NameError: name 'hrdaily_returns_std' is not defined
```

```
hrdaily_returns_std.isnull().any()
```

Professor's Strategies Daily Std

Professor's Strategies Daily Returns

```
print(professor_daily_returns.shape)
professor_daily_returns.head()

professor_daily_returns.isnull().any()

import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")
passive_strategy_std = dict(professor_daily_returns.std())

passive_strategy_std

np.random.seed(0)
random_target = float(np.random.rand(1))
random_target

WTplus1_prof = random_target / hrdaily_returns_std
WTplus1_prof.index = WTplus1_prof.index + pd.Timedelta(days=1)
WTplus1_prof = WTplus1_prof[:-1]
print(WTplus1_prof.shape)
WTplus1_prof

print(professor_daily_returns.shape)
WTplus1_prof.shape

estimated_return = WTplus1_prof * professor_daily_returns
estimated_return = estimated_return[1:]
estimated_return

estimated_return_std = dict(estimated_return.std())
estimated_return_std

strategies = estimated_return_std.keys()
strategies
```

```
from scipy.optimize import minimize

#targets = np.linspace(0,1,100)
#for target in targets:
my_targets={}
for strategy in strategies:

  # Example input data
  passive_strategy_ = passive_strategy_std[strategy]
  estimated_return_ = estimated_return_std[strategy]
  professor_daily_return_ = professor_daily_returns[strategy]

  # Define the function to minimize
  def objective_function(target):
      return abs(passive_strategy_ - estimated_return_)

  # Initial guess for the 'target'
  initial_guess = random.random()

  # Solve for the target using scipy.optimize
  result = minimize(objective_function, initial_guess, method='Nelder-
Mead')

  # Get the optimized target
  optimized_target = result.x[0]

  # Output the result
  my_targets[strategy]=optimized_target
  #print(f"Optimized Target: {optimized_target}")
  # print(f"Passive Strategy Std: {passive_strategy_}")
  # print(f"Estimated Return Std with Optimized Target:
{(optimized_target / estimated_return_) * daily_returns}")

my_targets

import numpy as np
from scipy.optimize import minimize

# Dictionary to store optimized targets
my_targets = {}
estimated_std = estimated_return_std
# Loop through each strategy
for strategy in strategies:
    # Extract relevant precomputed values
    passive_std  = passive_strategy_std[strategy]
    precomputed_std  = estimated_std[strategy]

    # Define the objective function
    def objective_function(target):
        # Adjust the precomputed standard deviation by scaling the
```

```
target
        adjusted_std = precomputed_std  * target
        return abs(passive_strategy_ - adjusted_std)

    # Precomputed initial guess
    initial_guess = 1.0  # Adjust this if initial_guess_target is
precomputed

    # Solve for the optimized target
    result = minimize(objective_function, initial_guess,
method='Nelder-Mead')
    optimized_target = result.x[0]

    # Store the result in the targets dictionary
    my_targets[strategy] = optimized_target

# Output the optimized targets
print("Optimized Targets for Strategies:")
print(my_targets)

# Verify the optimization result
for strategy, target in my_targets.items():
    print(f"\nStrategy: {strategy}")
    print(f"Passive Strategy Std: {passive_strategy_std[strategy]}")
    print(f"Precomputed Estimated Std: {estimated_std[strategy]}")
    print(f"Optimized Adjusted Std: {estimated_std[strategy] *
target}")
    print(f"Difference: {abs(passive_strategy_std[strategy] -
estimated_std[strategy] * target)}")

strategies = list(strategies)

strategies

WTplus1_ours = {}

for i in strategies:
 WTplus1_ours[i]= my_targets[i] / hrdaily_returns_std[i]

WTplus1_ours

WTplus1_ours=pd.DataFrame(WTplus1_ours)
WTplus1_ours.index = WTplus1_ours.index + pd.Timedelta(days=1)
WTplus1_ours = WTplus1_ours[:-1]
WTplus1_ours

professor_daily_returns.shape

our_returns = WTplus1_ours * professor_daily_returns
our_returns = our_returns[1:]
our_returns
```

```
# annualized return and volatility
freq = 'daily'
if freq == 'monthly':
    ann_mean = ((1+pf_ret.mean()) ** 12) - 1
    ann_vol = pf_ret.std() * (12**0.5)
elif freq == 'daily':
    ann_mean = ((1+pf_ret.mean()) ** 365) - 1
    ann_vol = pf_ret.std() * (365**0.5)

# Sharpe ratio
sharpe_ratio = ann_mean / ann_vol

# maximum drawdown
cummulative_return = (1 + pf_ret).cumprod()
previous_peaks = cummulative_return.cummax()
drawdown = (previous_peaks - cummulative_return)/previous_peaks
max_drawdown = abs(drawdown.max())

summ_tab_martina = pd.DataFrame(columns=['EW', 'FW', 'MV', 'meanvar',
'MD', 'RP', 'strategy'])
summ_tab_martina.loc['Annualized return'] = ann_mean
summ_tab_martina.loc['Annualized volatility'] = ann_vol
summ_tab_martina.loc['Sharpe ratio'] = sharpe_ratio
summ_tab_martina.loc['Max DD'] = max_drawdown

summ_tab_martina

# annualized return and volatility
freq = 'daily'
if freq == 'monthly':
    ann_mean = ((1+our_returns.mean()) ** 12) - 1
    ann_vol = our_returns.std() * (12**0.5)
elif freq == 'daily':
    ann_mean = ((1+our_returns.mean()) ** 365) - 1
    ann_vol = our_returns.std() * (365**0.5)

# Sharpe ratio
sharpe_ratio = ann_mean / ann_vol

# maximum drawdown
cummulative_return = (1 + our_returns).cumprod()
previous_peaks = cummulative_return.cummax()
drawdown = (previous_peaks - cummulative_return)/previous_peaks
max_drawdown = abs(drawdown.max())

summ_tab_strategy = pd.DataFrame(columns=['EW', 'FW', 'MV', 'meanvar',
'MD', 'RP'])
summ_tab_strategy.loc['Annualized return'] = ann_mean
```

```
summ_tab_strategy.loc['Annualized volatility'] = ann_vol
summ_tab_strategy.loc['Sharpe ratio'] = sharpe_ratio
summ_tab_strategy.loc['Max DD'] = max_drawdown

summ_tab_strategy

our_returns.to_csv("our_strategy_returns.csv")

our_return_mu = dict(our_returns.mean())
our_return_mu

professor_mu = dict(professor_daily_returns.mean())
professor_mu

final_ans = {}

for i in professor_mu.keys():
  final_ans[i]=our_return_mu[i]-professor_mu[i]
final_ans

pf_ret = our_returns.dropna()


plt.figure(figsize=(12, 12))
plt.plot((1+pf_ret['EW']).cumprod(), label='EW')
plt.plot((1+pf_ret['FW']).cumprod(), label='FW')
plt.plot((1+pf_ret['MV']).cumprod(), label='MV')
plt.plot((1+pf_ret['meanvar']).cumprod(), label='meanvar')
plt.plot((1+pf_ret['MD']).cumprod(), label='MD')
plt.plot((1+pf_ret['RP']).cumprod(), label='RP')
#plt.plot((1+pf_ret['mom']).cumprod(), label='mom')
plt.legend()
plt.title('Cumulative returns (without fees!)')
plt.show()

my_X = professor_daily_returns[1:].copy()
my_X

strategies

professor_daily_returns.head()

# Import required libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels.api as sm

# # Example dataset
# # Create a sample dataset
```

```
# #X = np.random.rand(100, 1) * 10  # 100 samples, single feature
# #y = 2.5 * X + np.random.randn(100, 1) * 2  # y = 2.5X + noise

# pingpong_res = {}
# for i in strategies:
#   X = my_X[i]
#   y = pf_ret[i]
#   # Split the dataset into training and testing sets
#   X_train, X_test, y_train, y_test = X[X.index < '2020-11-
03'].values.reshape(-1, 1), X[X.index >= '2020-11-
03'].values.reshape(-1, 1), y[y.index < '2020-11-03'].values.reshape(-
1, 1), y[y.index >= '2020-11-03'].values.reshape(-1, 1)
#train_test_split(X, y, test_size=0.2, random_state=42)


#   # Initialize the Linear Regression model
#   model = LinearRegression()

#   # Fit the model to the training data
#   model.fit(X_train, y_train)

#   # Predict on the test data
#   y_pred = model.predict(X_test)

#   # Evaluate the model
#   mse = mean_squared_error(y_test, y_pred)
#   r2 = r2_score(y_test, y_pred)

#   # Add a constant to X for the intercept term (needed for
statsmodels)
#   X_train_sm = sm.add_constant(X_train)

#   # Fit a linear regression model with statsmodels
#   ols_model = sm.OLS(y_train, X_train_sm).fit()

#   # Display the summary (includes p-values)
#   # print(ols_model.pvalues)

#   # print("Mean Squared Error:", mse)
#   # print("R-squared:", r2)

#   # # Model coefficients
#   # print("Coefficient:", model.coef_)
#   # print("Intercept:", model.intercept_)

#   pingpong_res[i] = {"Mean Squared Error": mse,
#                      "R-squared": r2,
#                      "Coefficient": model.coef_,
#                      "Intercept": model.intercept_,
#                      "p-values":ols_model.pvalues[0]}
```

```
#   print("p-values",ols_model.pvalues[0])
# # Split the data
# #X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# # # Fit a linear regression model with scikit-learn
# # model = LinearRegression()
# # model.fit(X_train, y_train)

# # # Add a constant to X for the intercept term (needed for
statsmodels)
# # X_train_sm = sm.add_constant(X_train)

# # # Fit a linear regression model with statsmodels
# # ols_model = sm.OLS(y_train, X_train_sm).fit()

# # # Display the summary (includes p-values)
# # print(ols_model.summary())
# pingpong_res
# pd.DataFrame(pingpong_res)
```

Hierarchical Risk Parity (HRP)

The HRP method works by finding subclusters of similar assets based on returns and constructing a hierarchy from these clusters to generate weights for each asset.

Here, we will use the max Sharpe statistic. The Sharpe ratio is the ratio between returns and risk. The lower the risk and the higher the returns, the higher the Sharpe ratio. The algorithm looks for the maximum Sharpe ratio, which translates to the portfolio with the highest return and lowest risk. Ultimately, the higher the Sharpe ratio, the better the performance of the portfolio.

```
from pypfopt import expected_returns

rets = expected_returns.returns_from_prices(daily_data)
rets.tail()
```


```
{"summary":"{\n  \"name\": \"rets\",\n  \"rows\": 5,\n  \"fields\": [\
n    {\n      \"column\": \"date\",\n      \"properties\": {\n
\"dtype\": \"date\",\n        \"min\": \"2024-11-27 00:00:00\",\n
\"max\": \"2024-12-01 00:00:00\",\n        \"num_unique_values\": 5,\n
\"samples\": [\n          \"2024-11-28 00:00:00\",\n          \"2024-
12-01 00:00:00\",\n          \"2024-11-29 00:00:00\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BTCUSD\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.02152512010900383,\n        \"min\": -0.010983263598326354,\n
\"max\": 0.044056217909668405,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          -0.003302840234220339,\n
```

```
0.008813678829543425,\n          0.019349578198011752\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ETHUSD\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.04725580147079744,\n        \"min\": -0.021487738866562744,\n
\"max\": 0.10078242551910921,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          -0.021487738866562744,\n
0.001025281277824197,\n          0.004190763557120114\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"XRPUSD\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.061887146430954325,\n        \"min\": 0.04848035984861365,\n
\"max\": 0.17564074345179903,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.04848035984861365,\n
0.17564074345179903,\n          0.1678590870207637\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ADAUSD\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.02501251729027812,\n        \"min\": 0.0046970190206017115,\n
\"max\": 0.06365487872616171,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.0169633445962587,\n
0.06365487872616171,\n          0.04145880757495024\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"DOGEUSD\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.029740962515651184,\n        \"min\": -0.0095488351359595,\n
\"max\": 0.06027363184079593,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.0014199237725132097,\n
0.04519613416714052,\n          0.06027363184079593\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"DOTUSD\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.024493146091088165,\n        \"min\": -0.010156767498344044,\n
\"max\": 0.0496709300881657,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.02176741985094055,\n
0.03022529556100828,\n          0.04874377677434283\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"LTCUSD\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.07979092309035121,\n        \"min\": -0.02145922746781115,\n
\"max\": 0.16627680311890836,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          -0.01839293053842983,\n
0.16627680311890836,\n          0.09756097560975596\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"SOLUSD\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.031873082416957806,\n        \"min\": -0.02425011973214486,\n
\"max\": 0.05068178134282886,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          -0.02005421820434783,\n          -
```

```
0.00326589975853786,\n          0.025688850387323825\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

```
mu = mean_historical_return(daily_data)
S = CovarianceShrinkage(daily_data).ledoit_wolf()

# ef = EfficientFrontier(mu, S) # Long-only
ef = EfficientFrontier(mu, S, weight_bounds=(-1,1)) #long-short
weights = ef.min_volatility()

cleaned_weights = ef.clean_weights()
print(dict(cleaned_weights))
```

```
{'BTCUSD': 0.80566, 'ETHUSD': 0.20885, 'XRPUSD': 0.09521, 'ADAUSD': -
0.05602, 'DOGEUSD': -0.12464, 'DOTUSD': 0.06758, 'LTCUSD': 0.12956,
'SOLUSD': -0.12621}
```

```
# display portfolio performance
ef.portfolio_performance(verbose=True)
```

```
Expected annual return: 55.1%
Annual volatility: 38.6%
Sharpe Ratio: 1.43

(0.551480745868802, 0.3856600007934029, 1.4299661482504349)
```

```
# Run the optimization algorithm to get the weights:
hrp = HRPOpt(rets)
hrp.optimize()
hrp_weights = hrp.clean_weights()
hrp_weights
```

```
OrderedDict([('BTCUSD', 0.23181),
             ('ETHUSD', 0.17645),
             ('XRPUSD', 0.08614),
             ('ADAUSD', 0.09885),
             ('DOGEUSD', 0.09021),
             ('DOTUSD', 0.09984),
             ('LTCUSD', 0.1396),
             ('SOLUSD', 0.07709)])
```

```
# the performance of the portfolio:
hrp.portfolio_performance(verbose=True)
```

```
Expected annual return: 67.8%
Annual volatility: 45.3%
Sharpe Ratio: 1.50

(0.678145729436125, 0.452744653427836, 1.4978547494746204)
```

```
# Compute weighted returns for each asset
idl_wgt_returns = rets.mul(hrp_weights, axis=1)
idl_wgt_returns.head()
```


```
{"summary":"{\n  \"name\": \"idl_wgt_returns\",\n  \"rows\": 710,\n
\"fields\": [\n    {\n      \"column\": \"date\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2022-12-23 00:00:00\",\n        \"max\": \"2024-12-01 00:00:00\",\n
\"num_unique_values\": 710,\n        \"samples\": [\n          \"2023-
10-13 00:00:00\",\n          \"2023-09-05 00:00:00\",\n
\"2024-11-24 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"BTCUSD\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.005927818904164927,\n        \"min\":
-0.019597207045982272,\n        \"max\": 0.027593025065293087,\n
\"num_unique_values\": 710,\n        \"samples\": [\n
0.0009616874719689,\n          -0.00032321778535186486,\n
0.0005500375351572686\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ETHUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0051717828480139436,\n        \"min\": -
0.01820488026588644,\n        \"max\": 0.0339695984890423,\n
\"num_unique_values\": 709,\n        \"samples\": [\n
0.0014440568978955548,\n          0.0004655672823218867,\n
0.005039172531630125\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"XRPUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.003938365473593091,\n        \"min\": -
0.011946284471479792,\n        \"max\": 0.06309264903192933,\n
\"num_unique_values\": 710,\n        \"samples\": [\n
0.0004048904625833383,\n          -0.0005455330029108665,\n          -
0.002147933886546046\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ADAUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0038692768406857823,\n        \"min\": -
0.013955075717480739,\n        \"max\": 0.023915710757886687,\n
\"num_unique_values\": 710,\n        \"samples\": [\n
0.0003106714285714257,\n          0.000571030444964885,\n          -
0.003890525930170282\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DOGEUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.004025681442730328,\n        \"min\": -
0.014593736678143952,\n        \"max\": 0.024152348993288587,\n
\"num_unique_values\": 706,\n        \"samples\": [\n
0.0031476437444844228,\n          -0.00530932929292929,\n
0.0008346443197330501\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DOTUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.003888744431831922,\n        \"min\": -
0.01318754404890328,\n        \"max\": 0.029356665652570734,\n
```

\"num_unique_values\": 710,\n   \"samples\": [\n
0.0014970556161395775,\n   0.000613094000944734,\n
0.0037673257317503185\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"LTCUSD\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0.0050137526724535995,\n   \"min\": -
0.018835292800358294,\n   \"max\": 0.03901556603773586,\n
\"num_unique_values\": 709,\n   \"samples\": [\n
0.0008913555992141577,\n   -0.0012927954795165664,\n   -
0.0058397084455712326\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"SOLUSD\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0.0037215412243200327,\n   \"min\": -
0.010256072421195287,\n   \"max\": 0.025762020620935137,\n
\"num_unique_values\": 710,\n   \"samples\": [\n
0.0019841003643201317,\n   0.003146029013231295,\n   -
0.0007511932121802412\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"idl_wgt_returns"}

```
# adding hrp to pf_ret
pf_ret['hrp'] = idl_wgt_returns.sum(axis = 1)

pf_ret.head()
```

{"summary":"{\n \"name\": \"pf_ret\",\n \"rows\": 591,\n
\"fields\": [\n   {\n   \"column\": \"date\",\n
\"properties\": {\n   \"dtype\": \"date\",\n   \"min\":
\"2023-04-21 00:00:00\",\n   \"max\": \"2024-12-01 00:00:00\",\n
\"num_unique_values\": 591,\n   \"samples\": [\n   \"2024-
09-07 00:00:00\",\n   \"2024-11-27 00:00:00\",\n
\"2023-11-02 00:00:00\"\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"EW\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.030300793854671035,\n   \"min\":
-0.10501091308083613,\n   \"max\": 0.18492226299221762,\n
\"num_unique_values\": 591,\n   \"samples\": [\n
0.01716262771057124,\n   0.054871833314135976,\n   -
0.006971708403715318\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"FW\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0.02515120425137443,\n   \"min\": -0.08895333031392846,\
n   \"max\": 0.11353857474175466,\n   \"num_unique_values\":
591,\n   \"samples\": [\n   0.006786829638976232,\n
0.05231933865718494,\n   -0.0146166364834659\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"MV\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0.024962172340578823,\n
\"min\": -0.08981630752932694,\n   \"max\": 0.11681584345292113,\
n   \"num_unique_values\": 591,\n   \"samples\": [\n

```
0.007351084312805385,\n          0.05773559608713896,\n          -
0.015747777160901956\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"meanvar\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.04090253242889013,\n        \"min\": -
0.12461458627784987,\n        \"max\": 0.29819590125017603,\n
\"num_unique_values\": 591,\n        \"samples\": [\n
0.01788938037298414,\n          0.05107592187089015,\n          -
0.018328910343751996\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.03244169183871434,\n        \"min\": -0.10852147862229264,\
n        \"max\": 0.2685102574715663,\n        \"num_unique_values\":
591,\n        \"samples\": [\n          0.011962253682505936,\n
0.05141474490971207,\n          -0.013045464926076297\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"RP\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.029376490378502627,\n
\"min\": -0.10308634866335649,\n        \"max\": 0.17353350071866458,\
n        \"num_unique_values\": 591,\n        \"samples\": [\n
0.015665612763466587,\n          0.05593953263326537,\n          -
0.008046329228774761\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"mom\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.13134528738399362,\n        \"min\": -0.6262844109688537,\n
\"max\": 2.7951881579801947,\n        \"num_unique_values\": 591,\n
\"samples\": [\n          0.031903145319604226,\n
0.13292067346563802,\n          -0.010161265749818174\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"kms\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.029760155769066914,\n
\"min\": -0.11270329359357259,\n        \"max\": 0.10509647495840138,\
n        \"num_unique_values\": 591,\n        \"samples\": [\n
0.017803057364829393,\n          0.05306389322791497,\n          -
0.009694881161622874\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"kms_rebalanced\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.07438690045788217,\n        \"min\": -
0.3592417517444558,\n        \"max\": 0.5512345431562686,\n
\"num_unique_values\": 591,\n        \"samples\": [\n
0.004985900256067009,\n          0.04885610891851696,\n
0.007245501869696294\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"hrp\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.02837740310260863,\n        \"min\": -0.10128387813341429,\
n        \"max\": 0.14499466464212485,\n        \"num_unique_values\":
591,\n        \"samples\": [\n          0.014420373550629719,\n
0.05700562201521563,\n          -0.009082276938484537\n        ],\n
```

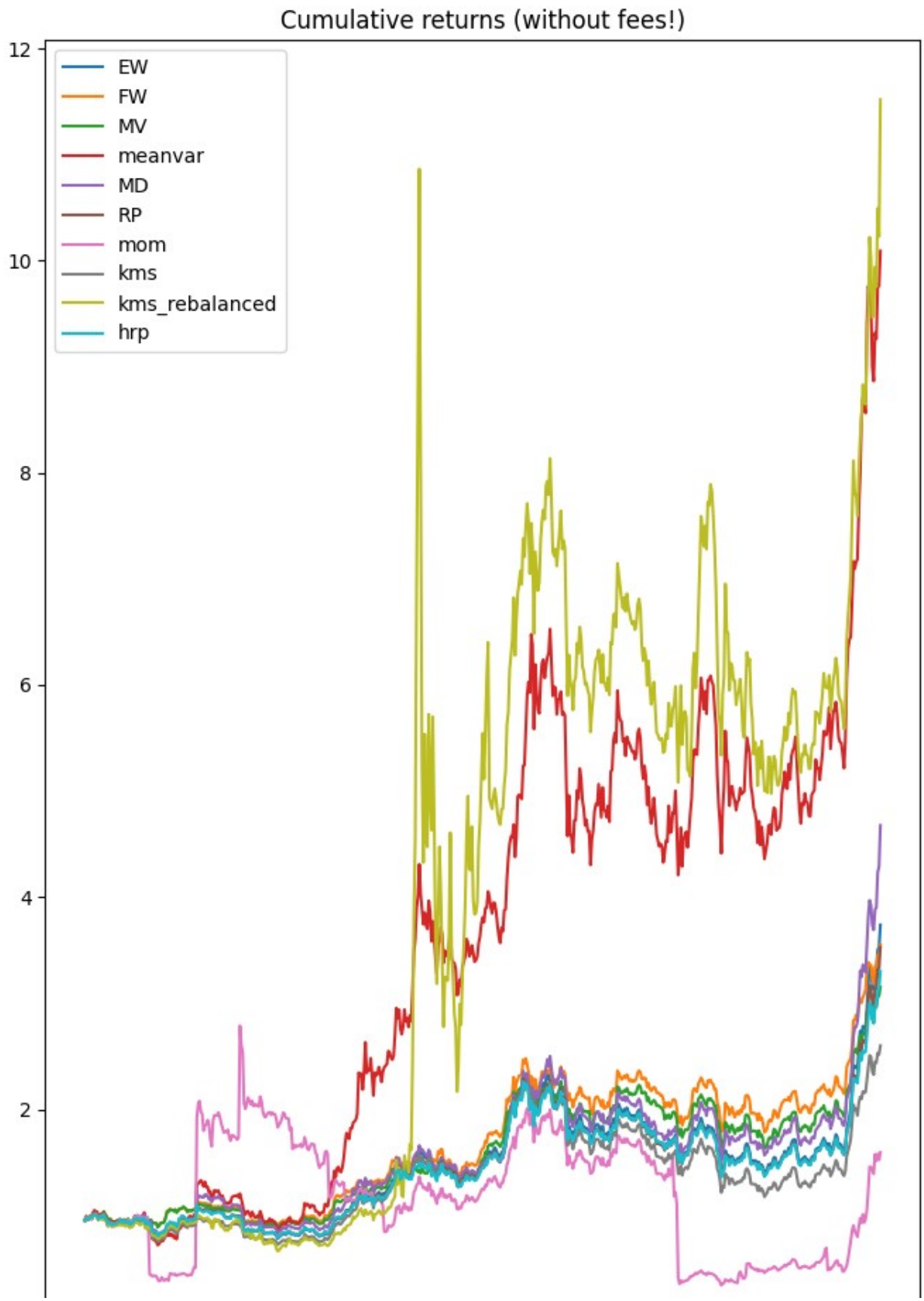
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"pf_ret"}

```
# annualized return and volatility
freq = 'daily'
if freq == 'monthly':
    ann_mean = ((1+pf_ret.mean()) ** 12) - 1
    ann_vol = pf_ret.std() * (12**0.5)
elif freq == 'daily':
    ann_mean = ((1+pf_ret.mean()) ** 365) - 1
    ann_vol = pf_ret.std() * (365**0.5)

# Sharpe ratio
sharpe_ratio = ann_mean / ann_vol

# maximum drawdown
cummulative_return = (1 + pf_ret).cumprod()
previous_peaks = cummulative_return.cummax()
drawdown = (previous_peaks - cummulative_return)/previous_peaks
max_drawdown = abs(drawdown.max())

summ_tab = pd.DataFrame(columns=['EW', 'FW', 'MV', 'meanvar', 'MD',
'RP', 'mom','kms','kms_rebalanced','hrp'])
summ_tab.loc['Annualized return'] = ann_mean
summ_tab.loc['Annualized volatility'] = ann_vol
summ_tab.loc['Sharpe ratio'] = sharpe_ratio
summ_tab.loc['Max DD'] = max_drawdown

summ_tab
```

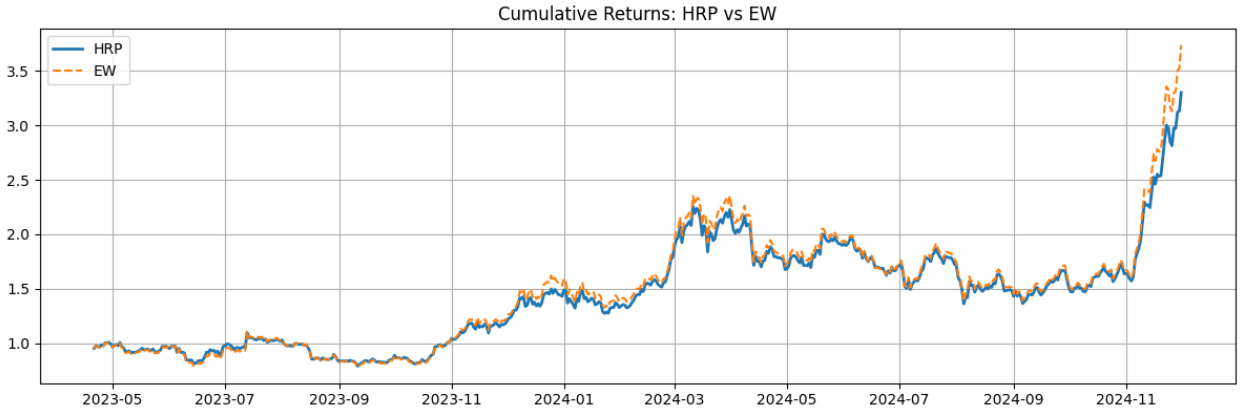
{"summary":"{\n \"name\": \"summ_tab\",\n \"rows\": 4,\n
\"fields\": [\n   {\n   \"column\": \"EW\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
1.1395962648726128,\n   \"min\": 0.41140709212069104,\n
\"max\": 2.875077305525959,\n   \"num_unique_values\": 4,\n
\"samples\": [\n   0.5788958537608414,\n
0.41140709212069104,\n   1.6643703314108693\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"FW\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1.251398058277812,\n
\"min\": 0.2810951226875451,\n   \"max\": 3.022045649940414,\n
\"num_unique_values\": 4,\n   \"samples\": [\n
0.48051308252995534,\n   0.2810951226875451,\n
1.4521324707991106\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"MV\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 1.0793705404245364,\n   \"min\": 0.3014373592709017,\n
\"max\": 2.670030432572351,\n   \"num_unique_values\": 4,\n
\"samples\": [\n   0.47690163294507265,\n
0.3014373592709017,\n   1.273341873306793\n   ],\n

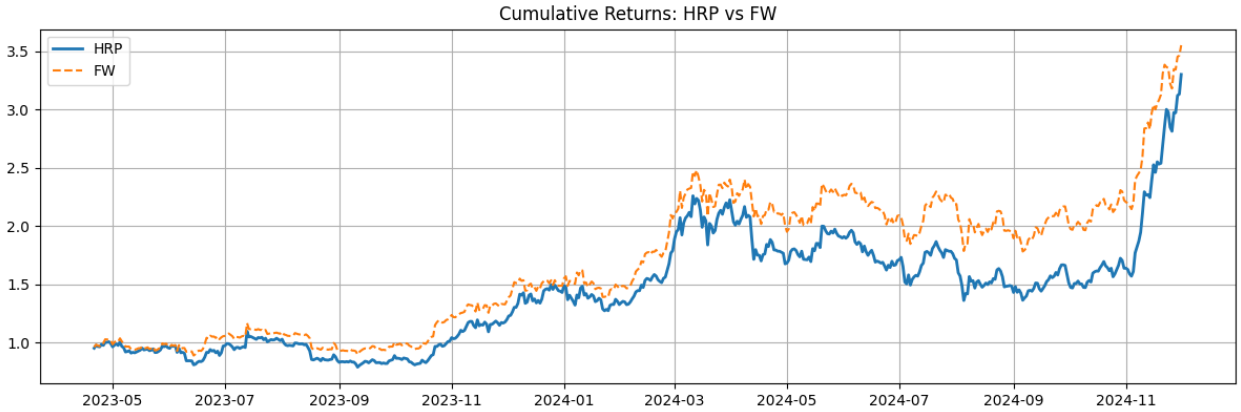
```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"meanvar\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
2.7601841482010974,\n        \"min\": 0.355388788082379,\n
\"max\": 5.896203324680888,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.7814417848248129,\n
0.355388788082379,\n          4.607539649728629\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"MD\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.4208021664175616,\n
\"min\": 0.39132375808570913,\n        \"max\": 3.4337897726733284,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.6197976523154216,\n          0.39132375808570913,\n
2.128254839647634\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"RP\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.0833625909365272,\n        \"min\": 0.4042561296208759,\n
\"max\": 2.753839813581291,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.5612370606435073,\n
0.4042561296208759,\n          1.545556962457428\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"mom\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2.4839774184972696,\n
\"min\": 0.8779213211391571,\n        \"max\": 6.716763936892997,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
2.509348192073813,\n          0.8779213211391571,\n
6.716763936892997\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"kms\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.6838018108967541,\n        \"min\": 0.48065634683156483,\n
\"max\": 1.9681867501730461,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.5685669776382385,\n
0.48065634683156483,\n          1.1190459919735156\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"kms_rebalanced\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4.734413789268068,\n        \"min\": 0.8005701092092389,\n
\"max\": 10.559359301371815,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          1.4211597377852243,\n
0.8005701092092389,\n          10.559359301371815\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"hrp\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.0207097487798458,\n
\"min\": 0.3974671573556025,\n        \"max\": 2.617220499974174,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.5421495250385254,\n          0.3974671573556025,\n
1.4189248509820906\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"summ_tab"}
```

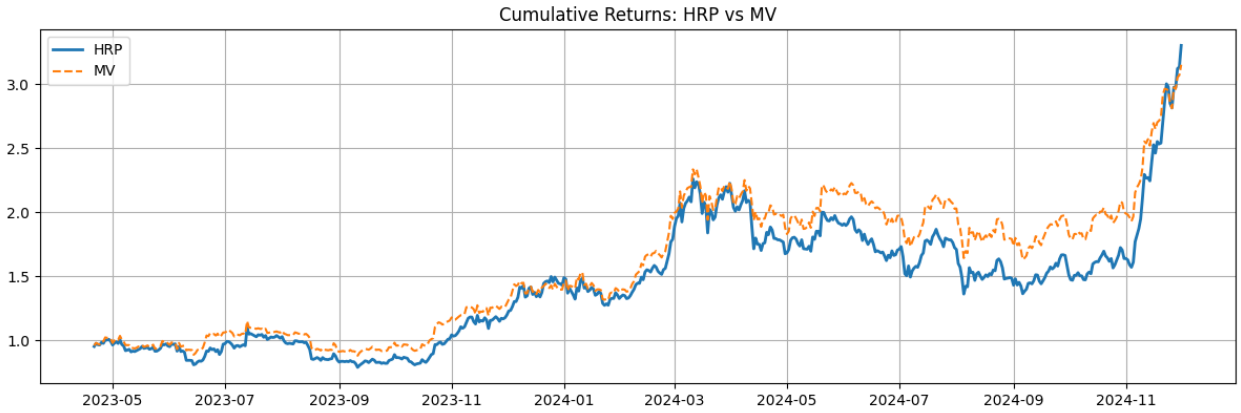
```
pf_ret = pf_ret.dropna()


plt.figure(figsize=(8, 12))
plt.plot((1+pf_ret['EW']).cumprod(), label='EW')
plt.plot((1+pf_ret['FW']).cumprod(), label='FW')
plt.plot((1+pf_ret['MV']).cumprod(), label='MV')
plt.plot((1+pf_ret['meanvar']).cumprod(), label='meanvar')
plt.plot((1+pf_ret['MD']).cumprod(), label='MD')
plt.plot((1+pf_ret['RP']).cumprod(), label='RP')
plt.plot((1+pf_ret['mom']).cumprod(), label='mom')
plt.plot((1+pf_ret['kms']).cumprod(), label='kms')
plt.plot((1+pf_ret['kms_rebalanced']).cumprod(),
label='kms_rebalanced')
plt.plot((1+pf_ret['hrp']).cumprod(), label='hrp')
plt.legend()
plt.title('Cumulative returns (without fees!)')
plt.show()
```

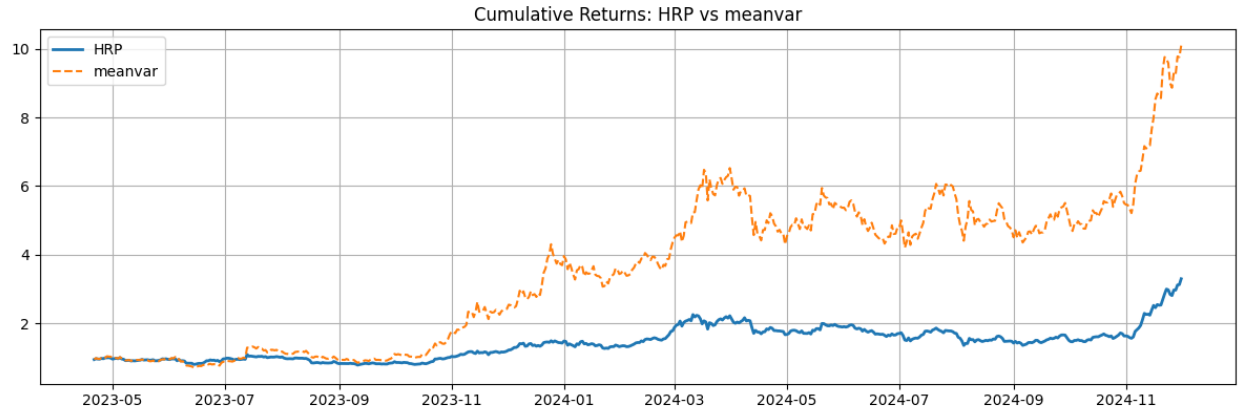

Cumulative returns (without fees!)
EW
FW
MV
meanvar
MD
RP
mom
kms
kms_rebalanced
hrp
12
10
8
6
4
2

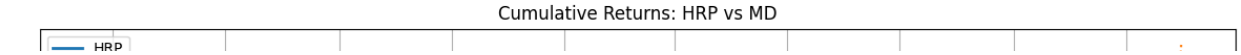
```
import matplotlib.pyplot as plt

# Drop missing values
pf_ret = pf_ret.dropna()

# List of all strategies
strategies = ['EW', 'FW', 'MV', 'meanvar', 'MD', 'RP', 'mom', 'kms',
'kms_rebalanced', 'hrp']

# Pair HRP with each strategy (excluding HRP vs. HRP)
hrp_combinations = [strategy for strategy in strategies if strategy !=
'hrp']

# Plot HRP combinations
plt.figure(figsize=(12, len(hrp_combinations) * 4))  # Adjust figure
size based on number of plots

for i, strategy in enumerate(hrp_combinations, 1):
    plt.subplot(len(hrp_combinations), 1, i)
    plt.plot((1 + pf_ret['hrp']).cumprod(), label='HRP',
linestyle='-', linewidth=2)
    plt.plot((1 + pf_ret[strategy]).cumprod(), label=strategy,
linestyle='--')
    plt.legend()
    plt.title(f'Cumulative Returns: HRP vs {strategy}')
    plt.grid()

plt.tight_layout()
plt.show()
```


Cumulative Returns: HRP vs EW
HRP
EW
3.5
3.0
2.5
2.0
1.5
1.0
2023-05
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11


Cumulative Returns: HRP vs FW
HRP
FW
3.5
3.0
2.5
2.0
1.5
1.0
2023-05
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11


Cumulative Returns: HRP vs MV
HRP
MV
3.0
2.5
2.0
1.5
1.0
2023-05
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11


Cumulative Returns: HRP vs meanvar
HRP
meanvar
10
8
6
4
2
2023-05
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11


Cumulative Returns: HRP vs MD
HRP

Kurtosis Minimization Strategy (KMS)

Reference: Analyzing Portfolio Optimization in Cryptocurrency Markets: A Comparative Study of Short-Term Investment Strategies Using Hourly Data Approach

Note: We tuned the strategy and applied KMS on daily data instead of hourly to match with existing strategies.

```
log_daily_returns = np.log(daily_data / daily_data.shift(1))
log_daily_returns = log_daily_returns[log_daily_returns.index >=
'2022-12-22'][1:]
log_daily_returns.head()
```


```
{"summary":"{\n  \"name\": \"log_daily_returns\",\n  \"rows\": 710,\n
\"fields\": [\n    {\n      \"column\": \"date\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2022-12-23 00:00:00\",\n        \"max\": \"2024-12-01 00:00:00\",\n
\"num_unique_values\": 710,\n        \"samples\": [\n          \"2023-
10-13 00:00:00\",\n          \"2023-09-05 00:00:00\",\n
\"2024-11-24 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"BTCUSD\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.025531403926562752,\n        \"min\":
-0.08832855900295963,\n        \"max\": 0.11246486297905127,\n
\"num_unique_values\": 710,\n        \"samples\": [\n
0.004140020459236567,\n          -0.0013952949823860783,\n
0.0023699840495238224\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ETHUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.029000576629994744,\n        \"min\": -0.1088923285354716,\
n        \"max\": 0.17606607511638206,\n        \"num_unique_values\":
709,\n        \"samples\": [\n          0.00815063701044738,\n
0.002635047638005032,\n          0.028158446331055634\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"XRPUSD\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.04114643272983057,\n        \"min\": -0.14929442942070006,\n
\"max\": 0.5495326114962481,\n        \"num_unique_values\": 710,\n
\"samples\": [\n          0.004689364581521381,\n          -
0.006353236451858923,\n          -0.02525153627374043\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ADAUSD\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.038385063472150996,\n        \"min\": -0.152189242744693,\n
\"max\": 0.21667419874894867,\n        \"num_unique_values\": 710,\n
\"samples\": [\n          0.003137928690927166,\n
0.005760115560309654,\n          -0.04015333776958046\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"DOGEUSD\",\n      \"properties\":
```

{\n   \"dtype\": \"number\",\n   \"std\":
0.04366125894318625,\n   \"min\": -0.17646890231378612,\n
\"max\": 0.23723162191466787,\n   \"num_unique_values\": 706,\n
\"samples\": [\n   0.034297463607416846,\n   -
0.060658292416953155,\n   0.009209697465122383\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"DOTUSD\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
0.038344922326896784,\n   \"min\": -0.14166354551325386,\n
\"max\": 0.2577668787517196,\n   \"num_unique_values\": 710,\n
\"samples\": [\n   0.014883240496361352,\n
0.006121987568605649,\n   0.03703913440074264\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"LTCUSD\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
0.03551449604463958,\n   \"min\": -0.14493710645765936,\n
\"max\": 0.2464546301826569,\n   \"num_unique_values\": 709,\n
\"samples\": [\n   0.0063647705684864815,\n   -
0.00930385959077807,\n   -0.042731861311026687\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"SOLUSD\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
0.04715181338130125,\n   \"min\": -0.14276272097874826,\n
\"max\": 0.28831769172489347,\n   \"num_unique_values\": 710,\n
\"samples\": [\n   0.02541182166334941,\n
0.03999908308989004,\n   -0.009792153511768988\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\
n}","type":"dataframe","variable_name":"log_daily_returns"}

```
# Calculate kurtosis for each asset
kurtosis = log_daily_returns.kurtosis()

# Calculate weights inversely proportional to kurtosis
weights = 1 / kurtosis
weights /= weights.sum()  # Normalize to sum to 1

print("Kurtosis:")
kurtosis
```

```
Kurtosis:

BTCUSD      2.419166
ETHUSD      3.858861
XRPUSD     47.638180
ADAUSD      4.581326
DOGEUSD     4.256658
DOTUSD      4.411806
LTCUSD      5.776183
```

```
SOLUSD      2.834171
dtype: float64
```

```
print("Portfolio Weights:")
weights
```

```
Portfolio Weights:
```

```
BTCUSD     0.217638
ETHUSD     0.136440
XRPUSD     0.011052
ADAUSD     0.114923
DOGEUSD    0.123689
DOTUSD     0.119339
LTCUSD     0.091150
SOLUSD     0.185769
dtype: float64
```

```
# Compute weighted returns for each asset
individual_weighted_returns = log_daily_returns.mul(weights, axis=1)

# Check individual contributions
print("Individual Asset Weighted Returns:")
individual_weighted_returns.head()

# Total portfolio return as a sum of all assets' weighted returns
# portfolio_returns = individual_weighted_returns.sum(axis=1) #
(weights * daily_returns).sum(axis=1)

# print("Total Portfolio Returns:")
# print(portfolio_returns.head())
```

```
Individual Asset Weighted Returns:
```


```
{"summary":"{\n  \"name\": \"# print(portfolio_returns\",\n  \"rows\":
5,\n  \"fields\": [\n    {\n      \"column\": \"date\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2022-12-23 00:00:00\",\n        \"max\": \"2022-12-27 00:00:00\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          \"2022-
12-24 00:00:00\",\n          \"2022-12-27 00:00:00\",\n
\"2022-12-25 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"BTCUSD\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.0015523178624787923,\n        \"min\":
-0.0028097686660824156,\n        \"max\": 0.001148009109818479,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.0007768614871116516,\n          -0.0028097686660824156,\n          -
0.00014221717082064697\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ETHUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0010787008183975118,\n        \"min\": -
```

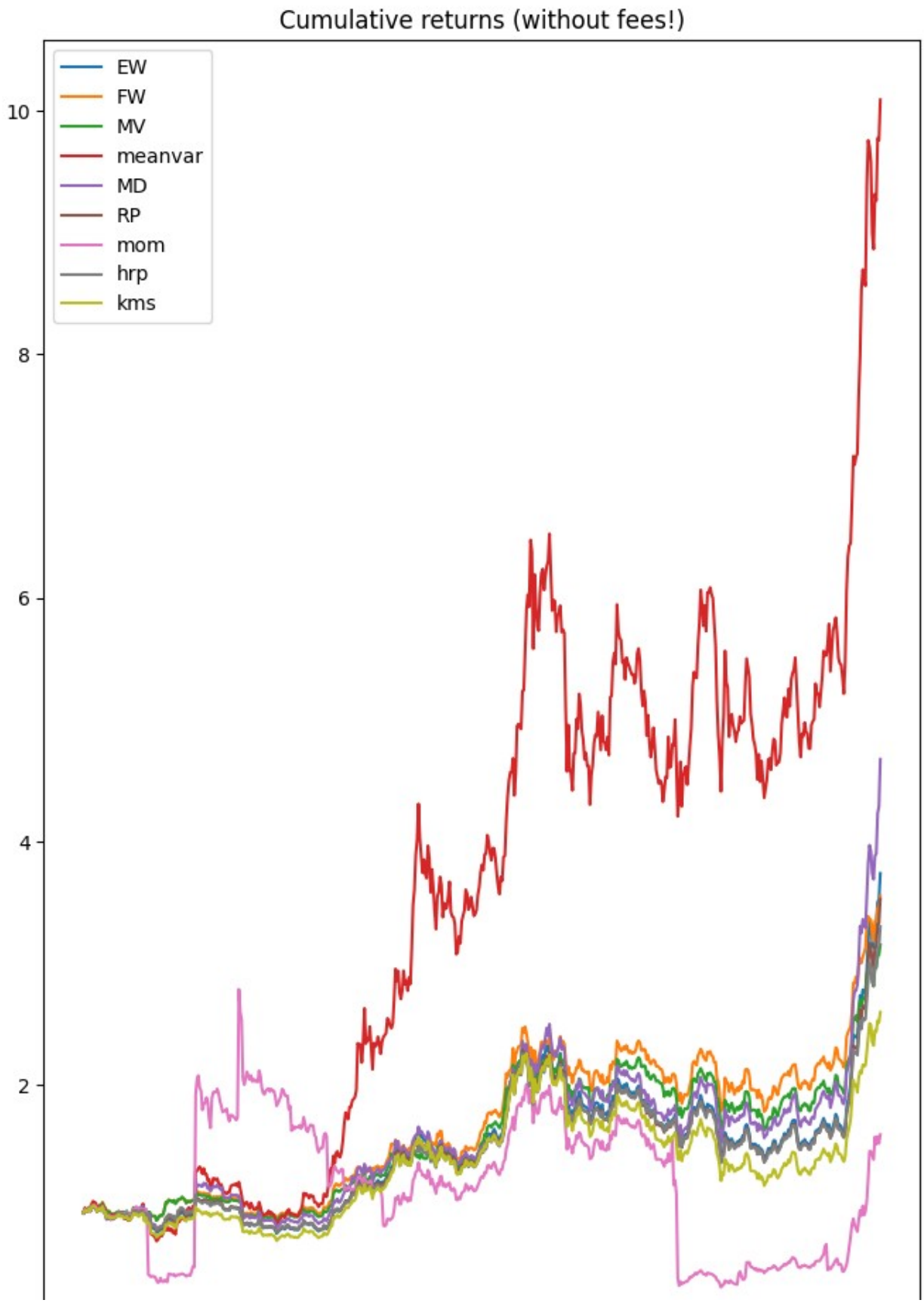
```
0.001868681051368173,\n        \"max\": 0.0010710822516962369,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
2.2359809658427818e-05,\n          -0.001868681051368173,\n          -
0.00026855986670730415\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"XRPUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.00032431328148016 47,\n        \"min\": -
0.00020437778764801338,\n        \"max\": 0.0006384160475168794,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          -
7.789911881247732e-05,\n          3.646512525864128e-05,\n          -
0.00020437778764801338\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ADAUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0014606371835433958,\n        \"min\": -
0.0018611295706830798,\n        \"max\": 0.0018212156237466105,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          -
0.00029279847502584243,\n          -0.0018611295706830798,\n
0.00020415041502841507\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DOGEUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.0016231401856600948,\n        \"min\": -
0.002455524522787507,\n        \"max\": 0.0014707912935227906,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.0014707912935227906,\n          -0.0015472751431432248,\n          -
0.00243951228469287\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DOTUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0020007314675607847,\n        \"min\": -
0.002424819485182013,\n        \"max\": 0.0027714646597904033,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          -
0.0006149450652278854,\n          -0.002424819485182013,\n
0.00045482985734896045\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"LTCUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0026606896569948864,\n        \"min\": -
0.0024468736235352816,\n        \"max\": 0.004568610643804616,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          -
2.77179209911859e-05,\n          -0.0024468736235352816,\n
0.004568610643804616\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"SOLUSD\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0030349739906504708,\n        \"min\": -
0.006166042414802603,\n        \"max\": 0.0006967265528260234,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          -
0.006166042414802603,\n          -0.004835884500402056,\n          -
0.00012556314924336054\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

#daily_asset_returns_hr =
individual_weighted_returns.groupby(pd.Grouper(freq='D')).mean() * 24
```

```
# Adding KMS to pf_ret
pf_ret['kms'] = individual_weighted_returns.sum(axis = 1)

# annualized return and volatility
freq = 'daily'
if freq == 'monthly':
    ann_mean = ((1+pf_ret.mean()) ** 12) - 1
    ann_vol = pf_ret.std() * (12**0.5)
elif freq == 'daily':
    ann_mean = ((1+pf_ret.mean()) ** 365) - 1
    ann_vol = pf_ret.std() * (365**0.5)

# Sharpe ratio
sharpe_ratio = ann_mean / ann_vol

# maximum drawdown
cummulative_return = (1 + pf_ret).cumprod()
previous_peaks = cummulative_return.cummax()
drawdown = (previous_peaks - cummulative_return)/previous_peaks
max_drawdown = abs(drawdown.max())

summ_tab = pd.DataFrame(columns=['EW', 'FW', 'MV', 'meanvar', 'MD',
'RP','kms']) # mom
summ_tab.loc['Annualized return'] = ann_mean
summ_tab.loc['Annualized volatility'] = ann_vol
summ_tab.loc['Sharpe ratio'] = sharpe_ratio
summ_tab.loc['Max DD'] = max_drawdown

summ_tab
```


```
{"summary":"{\n  \"name\": \"summ_tab\",\n  \"rows\": 4,\n
\"fields\": [\n    {\n      \"column\": \"EW\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.1395962648726128,\n        \"min\": 0.41140709212069104,\n
\"max\": 2.875077305525959,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.5788958537608414,\n
0.41140709212069104,\n          1.6643703314108693\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"FW\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.251398058277812,\n
\"min\": 0.2810951226875451,\n        \"max\": 3.022045649940414,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.48051308252995534,\n          0.2810951226875451,\n
1.4521324707991106\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MV\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.0793705404245364,\n        \"min\": 0.3014373592709017,\n
\"max\": 2.670030432572351,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.47690163294507265,\n
0.3014373592709017,\n          1.2733418733067 93\n        ],\n
```

\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"meanvar\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
2.7601841482010974,\n   \"min\": 0.355388788082379,\n
\"max\": 5.896203324680888,\n   \"num_unique_values\": 4,\n
\"samples\": [\n   0.7814417848248129,\n
0.355388788082379,\n   4.607539649728629\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"MD\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1.4208021664175616,\n
\"min\": 0.39132375808570913,\n   \"max\": 3.4337897726733284,\n
\"num_unique_values\": 4,\n   \"samples\": [\n
0.6197976523154216,\n   0.39132375808570913,\n
2.128254839647634\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"RP\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 1.0833625909365272,\n   \"min\": 0.4042561296208759,\n
\"max\": 2.753839813581291,\n   \"num_unique_values\": 4,\n
\"samples\": [\n   0.5612370606435073,\n
0.4042561296208759,\n   1.545556962457428\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"kms\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0.6838018108967541,\n
\"min\": 0.48065634683156483,\n   \"max\": 1.9681867501730461,\n
\"num_unique_values\": 4,\n   \"samples\": [\n
0.5685669776382385,\n   0.48065634683156483,\n
1.1190459919735156\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"summ_tab"}

```
pf_ret = pf_ret.dropna()


plt.figure(figsize=(8, 12))
plt.plot((1+pf_ret['EW']).cumprod(), label='EW')
plt.plot((1+pf_ret['FW']).cumprod(), label='FW')
plt.plot((1+pf_ret['MV']).cumprod(), label='MV')
plt.plot((1+pf_ret['meanvar']).cumprod(), label='meanvar')
plt.plot((1+pf_ret['MD']).cumprod(), label='MD')
plt.plot((1+pf_ret['RP']).cumprod(), label='RP')
plt.plot((1+pf_ret['mom']).cumprod(), label='mom')
plt.plot((1+pf_ret['hrp']).cumprod(), label='hrp')
plt.plot((1+pf_ret['kms']).cumprod(), label='kms')
plt.legend()
plt.title('Cumulative returns (without fees!)')
plt.show()
```


Cumulative returns (without fees!)
EW
FW
MV
meanvar
MD
RP
mom
hrp
kms
10
8
6
4
2

```python
# Define rebalancing period (e.g., weekly)
rebalancing_period = 70  # 5 trading days for weekly

# Initialize variables to store individual contributions
crypto_contributions = pd.DataFrame(index=daily_returns.index,
columns=daily_returns.columns).fillna(0)

for i in range(0, len(daily_returns), rebalancing_period):
    # Subset data for the current period
    period_data = daily_returns.iloc[i:i + rebalancing_period]

    # Recalculate weights
    period_kurtosis = period_data.kurtosis()
    period_weights = 1 / period_kurtosis
    period_weights /= period_weights.sum() # period_weights =
period_weights / sum(period_weights)

    # Calculate individual contributions for the period
    for asset in daily_returns.columns:
        crypto_contributions.loc[period_data.index, asset] =
period_weights[asset] * period_data[asset]

# Check individual contributions
print("Individual Contributions Over Time:")
print(crypto_contributions.head())
```

```
Individual Contributions Over Time:
              BTCUSD    ETHUSD    XRPUSD    ADAUSD   DOGEUSD    DOTUSD
\
date

2022-12-23  0.000058 -0.000160  0.006305 -0.003566  0.000437  0.000497

2022-12-24 -0.000083 -0.000014 -0.003153  0.000743 -0.000844  0.000207

2022-12-25  0.000015  0.000167 -0.008225 -0.000519  0.001377 -0.000153

2022-12-26 -0.000123 -0.000669  0.026696 -0.004663  0.001386 -0.000944

2022-12-27  0.000298  0.001155  0.001484  0.004690  0.000877  0.000808


              LTCUSD    SOLUSD
date
2022-12-23 -0.006014 -0.000050
2022-12-24 -0.000327  0.000436
2022-12-25  0.055263  0.000009
2022-12-26  0.022704  0.000070
2022-12-27 -0.028478  0.000343
```

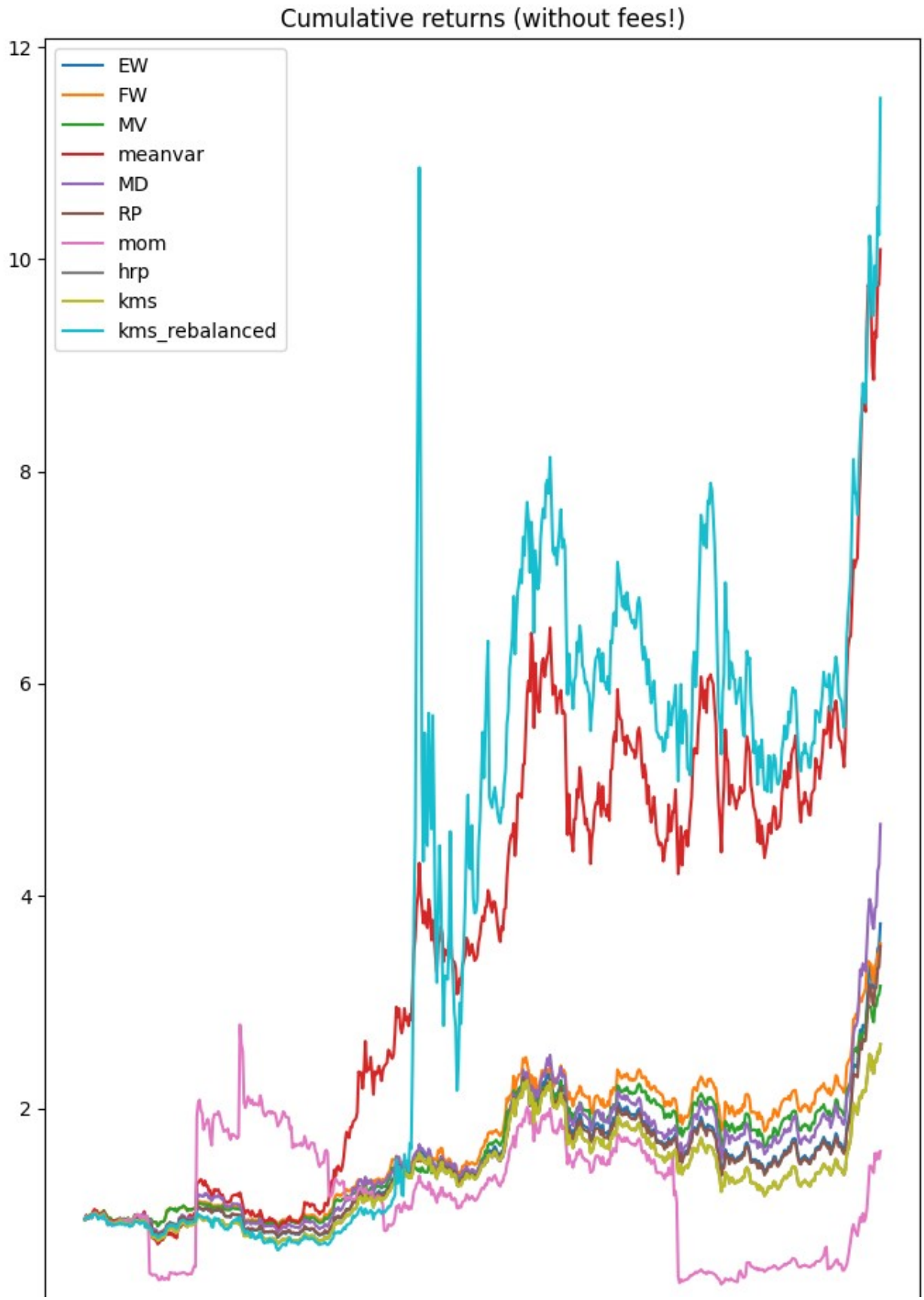
```
# Adding KMS to pf_ret
pf_ret['kms_rebalanced'] = crypto_contributions.sum(axis = 1)

# annualized return and volatility
freq = 'daily'
if freq == 'monthly':
    ann_mean = ((1+pf_ret.mean()) ** 12) - 1
    ann_vol = pf_ret.std() * (12**0.5)
elif freq == 'daily':
    ann_mean = ((1+pf_ret.mean()) ** 365) - 1
    ann_vol = pf_ret.std() * (365**0.5)

# Sharpe ratio
sharpe_ratio = ann_mean / ann_vol

# maximum drawdown
cummulative_return = (1 + pf_ret).cumprod()
previous_peaks = cummulative_return.cummax()
drawdown = (previous_peaks - cummulative_return)/previous_peaks
max_drawdown = abs(drawdown.max())

summ_tab = pd.DataFrame(columns=['EW', 'FW', 'MV', 'meanvar', 'MD',
'RP','kms','kms_rebalanced'])
summ_tab.loc['Annualized return'] = ann_mean
summ_tab.loc['Annualized volatility'] = ann_vol
summ_tab.loc['Sharpe ratio'] = sharpe_ratio
summ_tab.loc['Max DD'] = max_drawdown

summ_tab
```


```
{"summary":"{\n  \"name\": \"summ_tab\",\n  \"rows\": 4,\n
\"fields\": [\n    {\n      \"column\": \"EW\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.1395962648726128,\n        \"min\": 0.41140709212069104,\n
\"max\": 2.875077305525959,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.5788958537608414,\n
0.41140709212069104,\n          1.6643703314108693\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"FW\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.251398058277812,\n
\"min\": 0.2810951226875451,\n        \"max\": 3.022045649940414,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.48051308252995534,\n          0.2810951226875451,\n
1.4521324707991106\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MV\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.0793705404245364,\n        \"min\": 0.3014373592709017,\n
\"max\": 2.670030432572351,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.47690163294507265,\n
0.3014373592709017,\n          1.273341873306793\n        ],\n
```

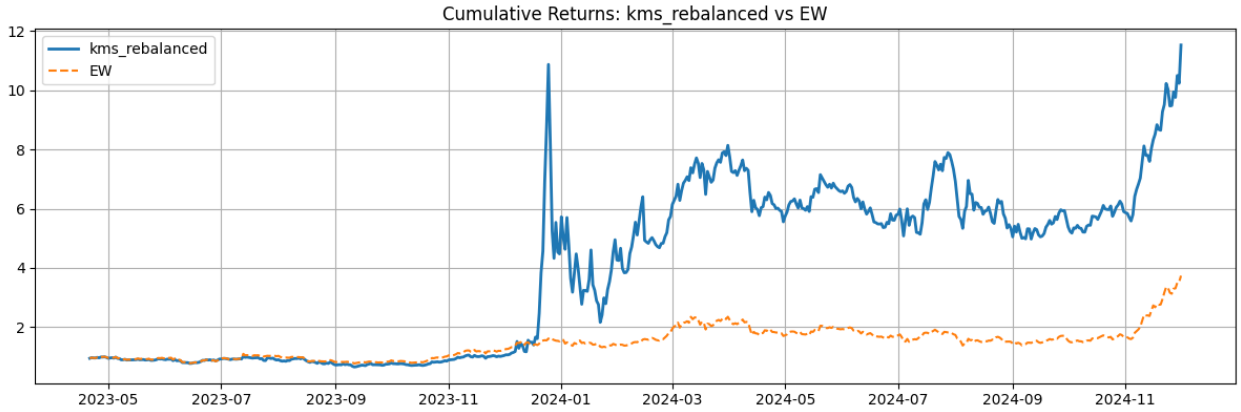
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"meanvar\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
2.7601841482010974,\n   \"min\": 0.355388788082379,\n
\"max\": 5.896203324680888,\n   \"num_unique_values\": 4,\n
\"samples\": [\n   0.7814417848248129,\n
0.355388788082379,\n   4.607539649728629\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"MD\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1.4208021664175616,\n
\"min\": 0.39132375808570913,\n   \"max\": 3.4337897726733284,\n
\"num_unique_values\": 4,\n   \"samples\": [\n
0.6197976523154216,\n   0.39132375808570913,\n
2.128254839647634\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"RP\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 1.0833625909365272,\n   \"min\": 0.4042561296208759,\n
\"max\": 2.753839813581291,\n   \"num_unique_values\": 4,\n
\"samples\": [\n   0.5612370606435073,\n
0.4042561296208759,\n   1.545556962457428\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"kms\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0.6838018108967541,\n
\"min\": 0.48065634683156483,\n   \"max\": 1.9681867501730461,\n
\"num_unique_values\": 4,\n   \"samples\": [\n
0.5685669776382385,\n   0.48065634683156483,\n
1.1190459919735156\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"kms_rebalanced\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 4.734413789268068,\n   \"min\":
0.8005701092092389,\n   \"max\": 10.559359301371815,\n
\"num_unique_values\": 4,\n   \"samples\": [\n
1.4211597377852243,\n   0.8005701092092389,\n
10.559359301371815\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"summ_tab"}

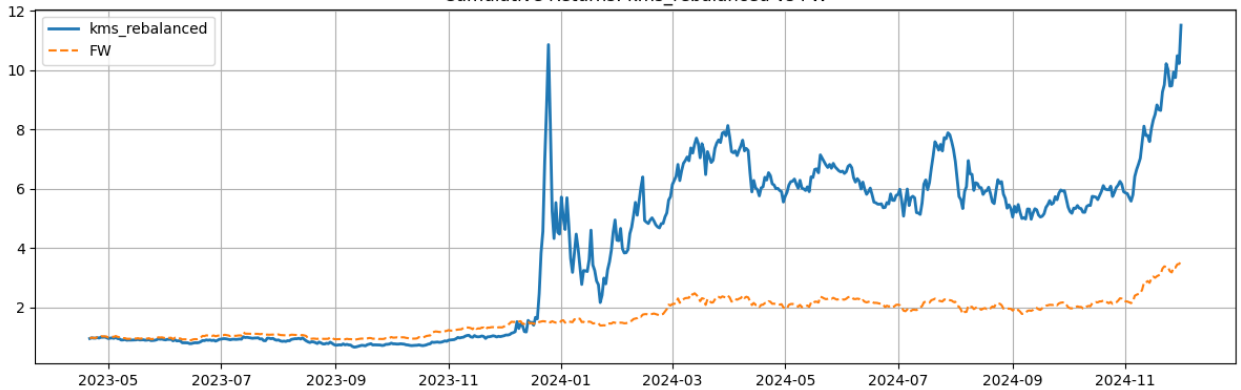
```
pf_ret = pf_ret.dropna()


plt.figure(figsize=(8, 12))
plt.plot((1+pf_ret['EW']).cumprod(), label='EW')
plt.plot((1+pf_ret['FW']).cumprod(), label='FW')
plt.plot((1+pf_ret['MV']).cumprod(), label='MV')
plt.plot((1+pf_ret['meanvar']).cumprod(), label='meanvar')
plt.plot((1+pf_ret['MD']).cumprod(), label='MD')
plt.plot((1+pf_ret['RP']).cumprod(), label='RP')
plt.plot((1+pf_ret['mom']).cumprod(), label='mom')
plt.plot((1+pf_ret['kms']).cumprod(), label='hrp')
plt.plot((1+pf_ret['kms']).cumprod(), label='kms')
```

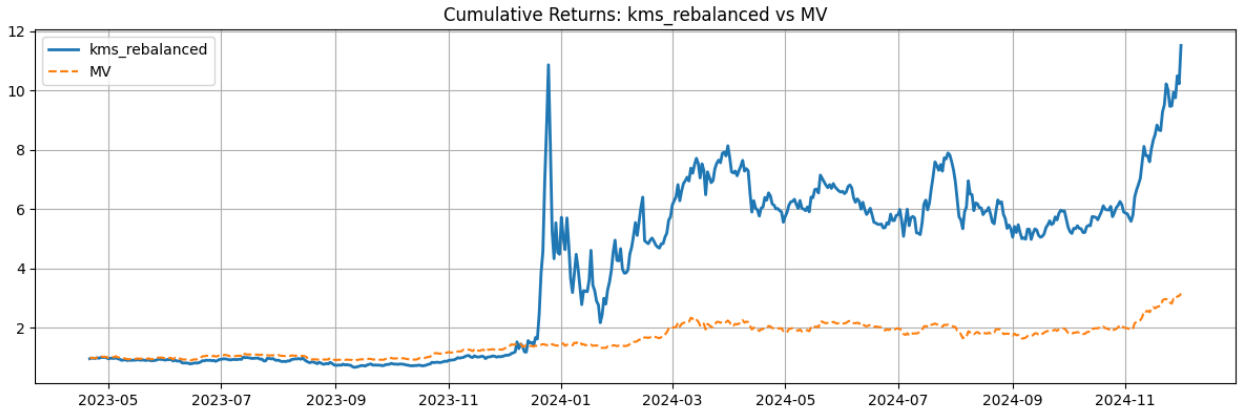
```
plt.plot((1+pf_ret['kms_rebalanced']).cumprod(),
label='kms_rebalanced')
plt.legend()
plt.title('Cumulative returns (without fees!)')
plt.show()
```

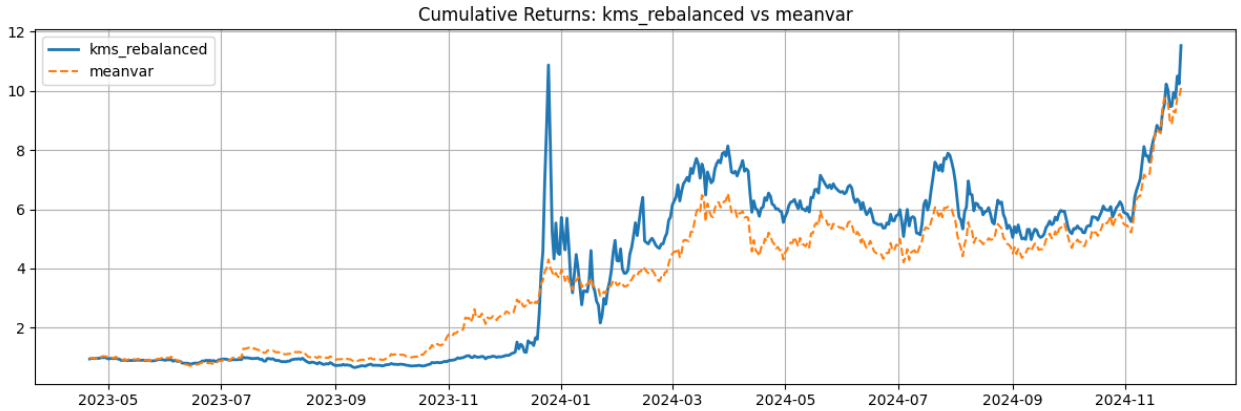

Cumulative returns (without fees!)
EW
FW
MV
meanvar
MD
RP
mom
hrp
kms
kms_rebalanced
12
10
8
6
4
2

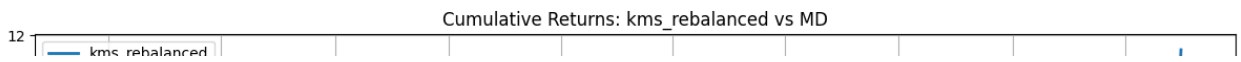
```
import matplotlib.pyplot as plt

# Drop missing values
pf_ret = pf_ret.dropna()

# List of all strategies
strategies = ['EW', 'FW', 'MV', 'meanvar', 'MD', 'RP', 'mom',
'hrp','kms', 'kms_rebalanced']

# Pair HRP with each strategy (excluding HRP vs. HRP)
kms_rebalanced_combinations = [strategy for strategy in strategies if
strategy != 'kms_rebalanced']

# Plot HRP combinations
plt.figure(figsize=(12, len(kms_rebalanced_combinations) * 4))  #
Adjust figure size based on number of plots

for i, strategy in enumerate(kms_rebalanced_combinations, 1):
    plt.subplot(len(kms_rebalanced_combinations), 1, i)
    plt.plot((1 + pf_ret['kms_rebalanced']).cumprod(),
label='kms_rebalanced', linestyle='-', linewidth=2)
    plt.plot((1 + pf_ret[strategy]).cumprod(), label=strategy,
linestyle='--')
    plt.legend()
    plt.title(f'Cumulative Returns: kms_rebalanced vs {strategy}')
    plt.grid()

plt.tight_layout()
plt.show()
```


Cumulative Returns: kms_rebalanced vs EW
kms_rebalanced
EW
12
10
8
6
4
2
2023-05
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11


Cumulative Returns: kms_rebalanced vs FW
kms_rebalanced
FW
12
10
8
6
4
2
2023-05
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11


Cumulative Returns: kms_rebalanced vs MV
kms_rebalanced
MV
12
10
8
6
4
2
2023-05
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11


Cumulative Returns: kms_rebalanced vs meanvar
kms_rebalanced
meanvar
12
10
8
6
4
2
2023-05
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11


Cumulative Returns: kms_rebalanced vs MD
12

HRP Scratch

```
import numpy as np
import pandas as pd
import scipy.cluster.hierarchy as sch
import scipy.spatial.distance as spdist

def preprocess_returns(returns, verbose=False):
    """
    Comprehensive returns preprocessing pipeline

    Args:
        returns (pd.DataFrame): Raw returns data
        verbose (bool): Print detailed preprocessing information

    Returns:
        pd.DataFrame: Cleaned and preprocessed returns
    """
    # Create a copy to avoid modifying original data
    processed_returns = returns.copy()

    # 1. Check for and handle missing values
    initial_nan_count = processed_returns.isna().sum()

    # Forward fill missing values (assumes time series data)
    processed_returns.fillna(method='ffill', inplace=True)

    # Backward fill any remaining missing values at the start
    processed_returns.fillna(method='bfill', inplace=True)

    # If still any NaNs, drop
    processed_returns.dropna(inplace=True)

    # 2. Winsorization to handle extreme values
    def winsorize(series, lower_quantile=0.01, upper_quantile=0.99):
        """
        Winsorize a series by capping at specified quantiles

        Args:
            series (pd.Series): Input series
            lower_quantile (float): Lower quantile threshold
            upper_quantile (float): Upper quantile threshold

        Returns:
            pd.Series: Winsorized series
        """
        lower_bound = series.quantile(lower_quantile)
        upper_bound = series.quantile(upper_quantile)
        return np.clip(series, lower_bound, upper_bound)
```

```
    # Apply winsorization to each column
    for col in processed_returns.columns:
        processed_returns[col] = winsorize(processed_returns[col])

    # 3. Remove any remaining NaN values after preprocessing
    processed_returns.dropna(inplace=True)

    # Verbose logging
    if verbose:
        print("Preprocessing Report:")
        print("-" * 20)
        print("Initial NaN counts:")
        print(initial_nan_count)
        print("\nFinal DataFrame shape:", processed_returns.shape)
        print("\nDescriptive Statistics:")
        print(processed_returns.describe())

    return processed_returns

preprocess_rets = preprocess_returns(daily_returns, verbose=False)

def hierarchical_risk_parity(returns, randomize_seed=42):
    """
    Implement Hierarchical Risk Parity portfolio allocation

    Args:
        returns (pd.DataFrame): Preprocessed asset returns
        randomize_seed (int): Random seed for reproducibility

    Returns:
        np.ndarray: HRP portfolio weights
    """
    # Set random seed for reproducibility
    np.random.seed(randomize_seed)

    # Compute correlation matrix
    corr_matrix = returns.corr()

    # Convert correlation matrix to distance matrix
    dist_matrix = np.sqrt(2 * (1 - corr_matrix))

    # Perform hierarchical clustering
    linkage_matrix = sch.linkage(spdist.squareform(dist_matrix),
method='ward')

    # Get the order of the clustering
    leaves_order = sch.leaves_list(linkage_matrix)

    # Reorder the correlation matrix and returns based on clustering
    reordered_corr = corr_matrix.iloc[leaves_order, leaves_order]
    reordered_returns = returns.iloc[:, leaves_order]
```

```
    def recursive_portfolio_weights(cov_matrix):
        """
        Recursively compute portfolio weights

        Args:
            cov_matrix (np.ndarray): Covariance matrix

        Returns:
            np.ndarray: Portfolio weights
        """
        # Base case
        if cov_matrix.shape[0] <= 2:
            return np.array([0.5, 0.5]) if cov_matrix.shape[0] == 2
else np.array([1.0])

        # Split the matrix
        mid = cov_matrix.shape[0] // 2
        left_cov = cov_matrix[:mid, :mid]
        right_cov = cov_matrix[mid:, mid:]

        # Recursive weights
        left_weights = recursive_portfolio_weights(left_cov)
        right_weights = recursive_portfolio_weights(right_cov)

        # Compute cluster variances
        def compute_cluster_variance(weights, cov_matrix):
            return np.dot(weights.T, np.dot(cov_matrix, weights))

        # Balance the two clusters
        left_var = compute_cluster_variance(left_weights, left_cov)
        right_var = compute_cluster_variance(right_weights, right_cov)

        # Adjust weights to equalize cluster risks
        total_var = left_var + right_var

        # Combine weights
        combined_weights = np.zeros(cov_matrix.shape[0])
        combined_weights[:mid] = left_weights * np.sqrt(total_var / (2
* left_var))
        combined_weights[mid:] = right_weights * np.sqrt(total_var /
(2 * right_var))

        return combined_weights

    # Compute HRP weights based on covariance matrix
    hrp_weights = recursive_portfolio_weights(reordered_corr.values)

    # Reverse the weights to match original asset order
    reverse_order = np.argsort(leaves_order)
```

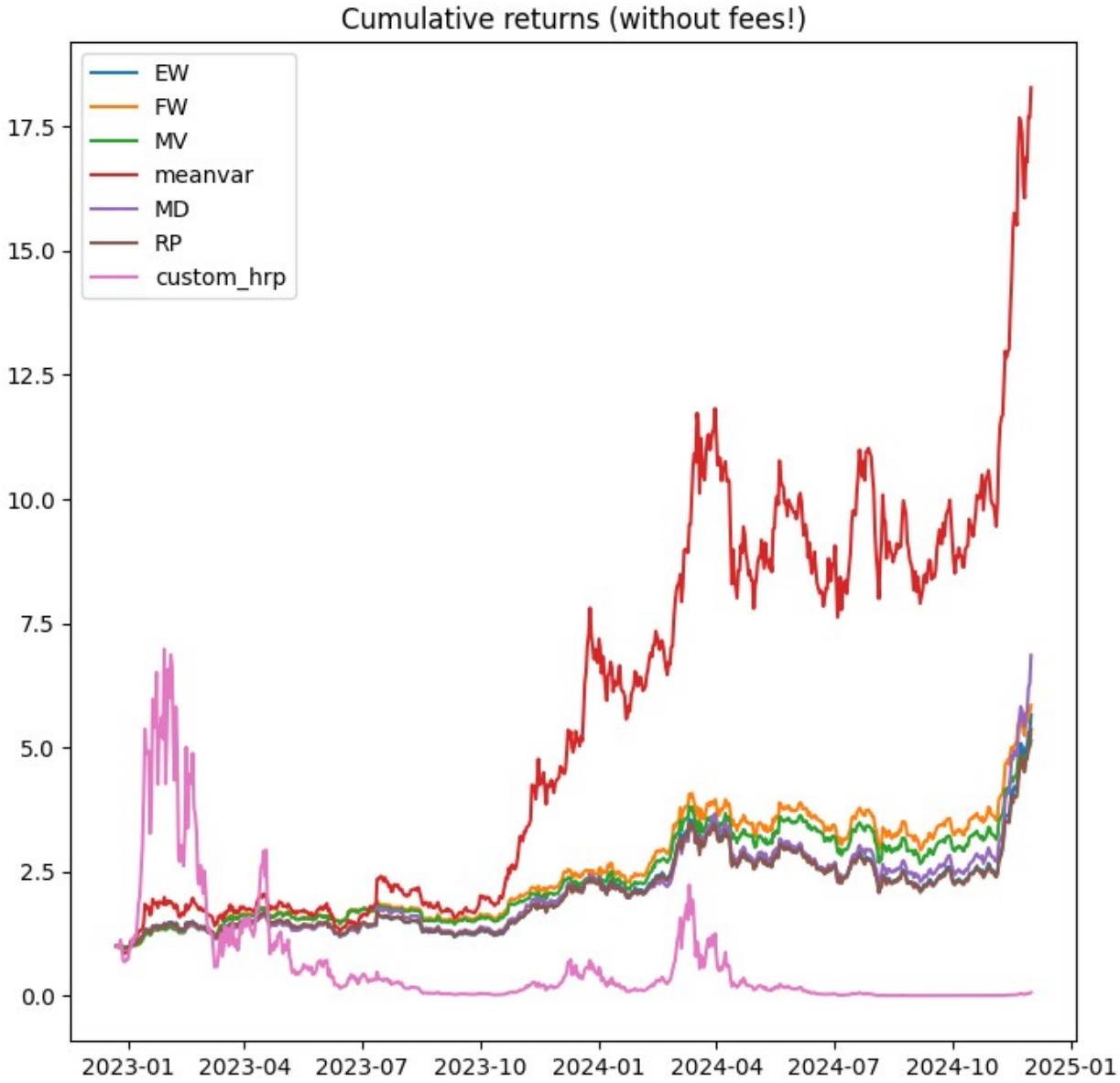
```
    final_weights = hrp_weights[reverse_order]

    # Normalize weights to sum to 1
    final_weights = final_weights / np.sum(final_weights)

    return final_weights

wgts = hierarchical_risk_parity(preprocess_rets, randomize_seed=42)

pf_ret['hrp'] = preprocess_rets.dot(wgts)

# annualized return and volatility
freq = 'daily'
if freq == 'monthly':
    ann_mean = ((1+pf_ret.mean()) ** 12) - 1
    ann_vol = pf_ret.std() * (12**0.5)
elif freq == 'daily':
    ann_mean = ((1+pf_ret.mean()) ** 365) - 1
    ann_vol = pf_ret.std() * (365**0.5)

# Sharpe ratio
sharpe_ratio = ann_mean / ann_vol

# maximum drawdown
cummulative_return = (1 + portfolio_return).cumprod()
previous_peaks = cummulative_return.cummax()
drawdown = (previous_peaks - cummulative_return)/previous_peaks
max_drawdown = abs(drawdown.max())

summ_tab_martina = pd.DataFrame(columns=['EW', 'FW', 'MV', 'meanvar',
'MD', 'RP', 'mom','kms','hrp'])
summ_tab_martina.loc['Annualized return'] = ann_mean
summ_tab_martina.loc['Annualized volatility'] = ann_vol
summ_tab_martina.loc['Sharpe ratio'] = sharpe_ratio
summ_tab_martina.loc['Max DD'] = max_drawdown

summ_tab_martina
```


```
{"summary":"{\n  \"name\": \"summ_tab_martina\",\n  \"rows\": 4,\n
\"fields\": [\n    {\n      \"column\": \"EW\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.1739418233845178,\n        \"min\": 0.5805446100524683,\n
\"max\": 3.2367333074574622,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.5805446100524683,\n
0.9997706460618814,\n          1.8790680758217282\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"FW\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.4005460176319608,\n
\"min\": 0.48482871421798895,\n        \"max\": 3.6788992417277053,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.48482871421798895,\n          0.9997706460618814,\n
```

```
1.7836359891043778\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MV\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.2375878704122447,\n        \"min\": 0.4810501545582174,\n
\"max\": 3.3270101755633843,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.4810501545582174,\n
0.9997706460618814,\n          1.600458759171528\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"meanvar\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
2.821645755509175,\n        \"min\": 0.818669366727281,\n
\"max\": 6.298629254653072,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.818669366727281,\n
0.9997706460618814,\n          5.1564948231567564\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"MD\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.3617521843488638,\n
\"min\": 0.6183846753601383,\n        \"max\": 3.6303224517132744,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.6183846753601383,\n          0.9997706460618814,\n
2.2449357707553346\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"RP\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.1353510055236349,\n        \"min\": 0.5630088667559382,\n
\"max\": 3.150218496182209,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.5630088667559382,\n
0.9997706460618814,\n          1.7736009455691413\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"mom\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\":
0.9997706460618814,\n        \"max\": 0.9997706460618814,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
0.9997706460618814\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"kms\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": null,\n        \"min\": 0.9997706460618814,\n        \"max\":
0.9997706460618814,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          0.9997706460618814\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"hrp\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.9400401451952817,\n
\"min\": 0.5397104081225921,\n        \"max\": 2.7229735043448295,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.5397104081225921\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"summ_tab_martina"}
```


```
pf_ret = pf_ret.dropna()
```

```
plt.figure(figsize=(8, 8))
plt.plot((1+pf_ret['EW']).cumprod(), label='EW')
plt.plot((1+pf_ret['FW']).cumprod(), label='FW')
plt.plot((1+pf_ret['MV']).cumprod(), label='MV')
plt.plot((1+pf_ret['meanvar']).cumprod(), label='meanvar')
plt.plot((1+pf_ret['MD']).cumprod(), label='MD')
plt.plot((1+pf_ret['RP']).cumprod(), label='RP')
plt.plot((1+pf_ret['custom_hrp']).cumprod(), label='custom_hrp')
#plt.plot((1+pf_ret['mom']).cumprod(), label='mom')
plt.legend()
plt.title('Cumulative returns (without fees!)')
plt.show()
```



```
hrp = calculate_portfolio_metrics(preprocessed_rets, wgts)
hrp
```

```
{'Return': 6.042907276654585,
 'Volatility': 3.803514179098832,
 'Sharpe Ratio': 1.588769488454052}
```