

MasterThesisCode

July 17, 2025

Author: Vojtěch Novotný

Numerical study code related to **Multi-objective portfolio optimization** thesis

1 Data

Our first task will be to load and examine daily returns of ten companies from S&P500.

```
[1]: # Importing libraries
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import yfinance as yf
from scipy.optimize import minimize
import seaborn as sns
import cvxpy as cp
import warnings
from tqdm import tqdm
import itertools
from matplotlib.collections import LineCollection
from matplotlib import cm
from matplotlib.ticker import StrMethodFormatter
from matplotlib.ticker import MaxNLocator
```

1.1 Data preprocessing

First, we download opening and closing prices of analyzed stocks from January 1st, 2015 till December 10th, 2024 and calculate the daily returns as $(1 + \frac{\text{Closing price}_t}{\text{Closing price}_{t-1}}) * 100\%$. We are only operating in trading days, and we check that we have all stock data available. To mitigate issues stemming from any missing data, we omit days if there is stock data missing.

```
[2]: # Define Constants
DATA_DIR = r'D:\Projects\ds2-hw\MasterThesis\stock_data' #needs to be changed
↳to correct local directory
START_DATE = '2015-01-01'
END_DATE = '2024-12-10'
STOCK_FILES = {
```

```

'AAPL': 'Apple Inc. Stock Price History.csv', #Information Technology 3
'AMZN': 'Amazon Stock Price History.csv', #Consumer Discretionary 4
'BAC': 'Bank of America Stock Price History.csv', #Financials 23
'COST': 'Costco Stock Price History.csv', #Consumer Staples 19
'IBM': 'IBM Stock Price History.csv', #Information Technology 31
'JNJ': 'Johnson & Johnson Stock Price History.csv', #Health Care 21
'JPM': 'JPMorgan Chase Stock Price History.csv', #Financials 11
'KO': 'The Coca-Cola Company Stock Price History.csv', #Consumer Staples 26
'T': 'AT&T Stock Price History.csv', #Communication Services 46
'UNH': 'UnitedHealth Group Stock Price History.csv', #Health Care 28
}

```

```

[3]: # Data loading function
def load_stock_data(stock_files, start_date, end_date, output_dir):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    for ticker, file in stock_files.items():
        stock = yf.Ticker(ticker)
        df = stock.history(start=start_date, end=end_date, auto_adjust=True)
        df.index = df.index.date
        df.index.name = 'Date'
        output_path = os.path.join(output_dir, file)
        df.to_csv(output_path)

```

```

[4]: # Load/download
load_stock_data(STOCK_FILES, START_DATE, END_DATE, DATA_DIR) #needs correctly
↳adjusted file names in the first code cell of the chapter

```

```

[5]: # Calculate Daily Returns
def calculate_daily_returns(data):
    data['Daily_Return'] = 100*data['Close'].pct_change()
    return data[['Date', 'Daily_Return']]

```

```

[6]: # Function that finds trading days with missing returns
def clean_dates(stock_returns):

    dropped_dates = stock_returns.index[stock_returns.isna().any(axis=1)].
↳tolist()

    cleaned = stock_returns.dropna(axis=0, how='any')

    valid_days = len(cleaned)

    return cleaned, valid_days, dropped_dates

```

```

[7]: # Load Stock Data and Calculate Daily Returns
daily_returns = {}

```

```

all_data = {}
for ticker, file_name in STOCK_FILES.items():
    file_path = os.path.join(DATA_DIR, file_name)
    data = pd.read_csv(file_path, parse_dates=['Date'])
    data['Ticker'] = ticker
    daily_returns[ticker] = calculate_daily_returns(data).set_index('Date')
    all_data[ticker] = data

stock_returns = pd.concat([df['Daily_Return'].rename(ticker) for ticker, df in
↳daily_returns.items()], axis=1)

```

```

[8]: # Clean the Data, should only delete the first day, since its return cannot be
↳computed
clean_returns, valid_days, dropped_dates = clean_dates(stock_returns)

print(f"{len(dropped_dates)} days omitted due to missing returns; {valid_days}
↳days remain.")

print(f"{dropped_dates} was/were dropped.") #always omits first day since we
↳cannot calculate return
stock_returns = stock_returns[~stock_returns.index.isin(dropped_dates)]

```

1 days omitted due to missing returns; 2500 days remain.
[Timestamp('2015-01-02 00:00:00')] was/were dropped.

1.2 Exploratory Analysis

Once we loaded and prepared all of our necessary data, we will want to shortly analyze what we have before we proceed to creating our portfolios.

```

[9]: mean_returns = stock_returns.mean()
cov_matrix = stock_returns.cov()
auto_cov = stock_returns.apply(lambda col: col.autocorr())

```

```

[10]: # Print basic info to know what to expect
print("\nMean:")
print(mean_returns)
print("\nCovariance:")
print(cov_matrix)
print("\nAuto-correlation:")
print(auto_cov)

```

Mean:

AAPL	0.108813
AMZN	0.128696
BAC	0.065259
COST	0.095437

```

IBM      0.044387
JNJ      0.031869
JPM      0.080131
KO       0.034881
T        0.036349
UNH      0.088239
dtype: float64

```

Covariance:

	AAPL	AMZN	BAC	COST	IBM	JNJ	JPM	\
AAPL	3.227022	2.053779	1.438923	1.183549	1.070768	0.705064	1.287864	
AMZN	2.053779	4.271864	1.226665	1.214267	0.885511	0.518660	1.042218	
BAC	1.438923	1.226665	3.858294	0.801793	1.539978	0.822069	3.028334	
COST	1.183549	1.214267	0.801793	1.852109	0.720824	0.583394	0.751206	
IBM	1.070768	0.885511	1.539978	0.720824	2.254734	0.731851	1.390834	
JNJ	0.705064	0.518660	0.822069	0.583394	0.731851	1.294897	0.785750	
JPM	1.287864	1.042218	3.028334	0.751206	1.390834	0.785750	2.981475	
KO	0.741274	0.504349	0.901400	0.628354	0.790475	0.655789	0.878216	
T	0.727926	0.530680	1.255209	0.519485	0.970891	0.650523	1.113885	
UNH	1.171454	0.891426	1.365265	0.787315	0.933124	0.861619	1.244916	

	KO	T	UNH
AAPL	0.741274	0.727926	1.171454
AMZN	0.504349	0.530680	0.891426
BAC	0.901400	1.255209	1.365265
COST	0.628354	0.519485	0.787315
IBM	0.790475	0.970891	0.933124
JNJ	0.655789	0.650523	0.861619
JPM	0.878216	1.113885	1.244916
KO	1.258562	0.747441	0.769523
T	0.747441	2.050486	0.775505
UNH	0.769523	0.775505	2.693094

Auto-correlation:

```

AAPL    -0.067732
AMZN    -0.011373
BAC     -0.043871
COST    -0.034809
IBM     -0.041762
JNJ     -0.077916
JPM     -0.101525
KO      -0.026072
T       -0.048779
UNH     -0.077549
dtype: float64

```

```

[11]: # Plot cumulative returns
cumulative_returns = (1 + stock_returns/100).cumprod()
split_date = pd.to_datetime("2022-12-12")

plt.figure(figsize=(8, 4))

# All assets
ax=cumulative_returns.plot(linewidth=0.8)
plt.axhline(y=1, color='gray', linestyle='--', linewidth=0.5)

ax.axvline(split_date, color='dimgray', linestyle='--', linewidth=1)
ax.axvspan(split_date + pd.Timedelta(days=1), cumulative_returns.index[-1],
           color='lightgray', alpha=0.3)

ax.set_ylabel("Relative Asset Value")
ax.set_xlabel("")
ax.tick_params(axis='x', labelsize=8, rotation=0)
ax.tick_params(axis='y', labelsize=8)

for label in ax.get_xticklabels():
    label.set_horizontalalignment('center')

ax.grid(True)
legend = ax.legend(title="Stock Tickers", loc="upper left", ncol=2)
for line in legend.get_lines():
    line.set_linewidth(2.0)

y_max = cumulative_returns.max().max()
y_text = y_max

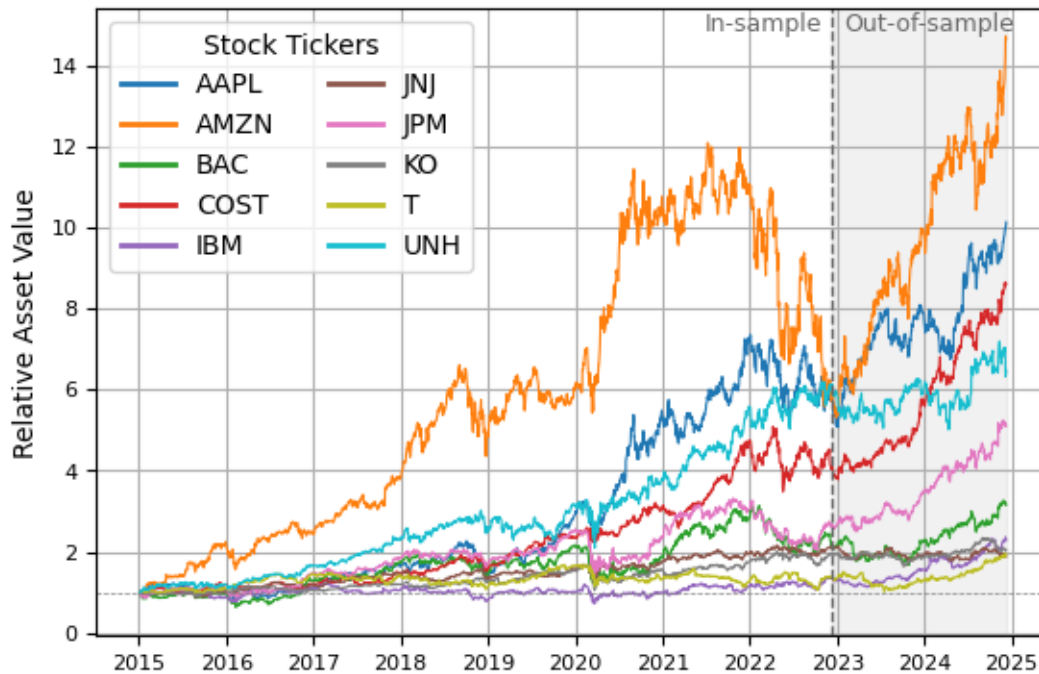
ax.text(split_date - pd.Timedelta(days=40), y_text, "In-sample",
        ha='right', va='bottom', fontsize=9, color='dimgray')

ax.text(split_date + pd.Timedelta(days=50), y_text, "Out-of-sample",
        ha='left', va='bottom', fontsize=9, color='dimgray')

plt.savefig("returns.pdf", dpi=300, bbox_inches='tight', transparent=True)
plt.show()

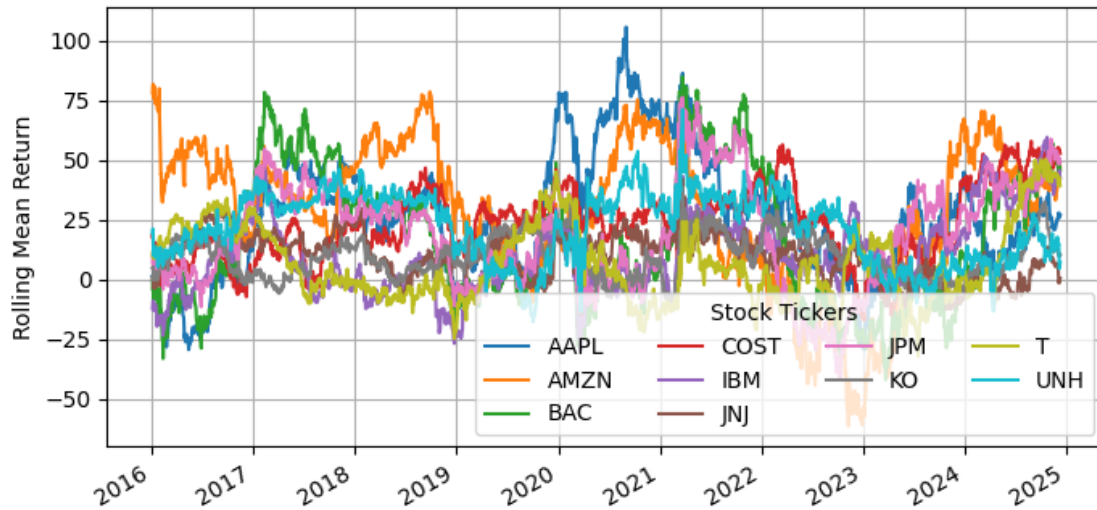
```

<Figure size 800x400 with 0 Axes>



```
[12]: # Plot rolling mean returns
rolling_window = 252 # one year
rolling_mean_returns = stock_returns.rolling(window=rolling_window).mean()*252

fig, ax = plt.subplots(figsize=(8, 4))
rolling_mean_returns.plot(ax=ax)
plt.ylabel("Rolling Mean Return")
plt.xlabel("")
plt.grid(True)
plt.legend(title="Stock Tickers", loc="lower right", bbox_to_anchor=(1, 0),
           ↪ncol=4)
plt.figtext(0.5,-0.005,"1-Year rolling yearly mean return.", ha="center",
           ↪fontsize=12)
plt.show()
```



1-Year rolling yearly mean return.

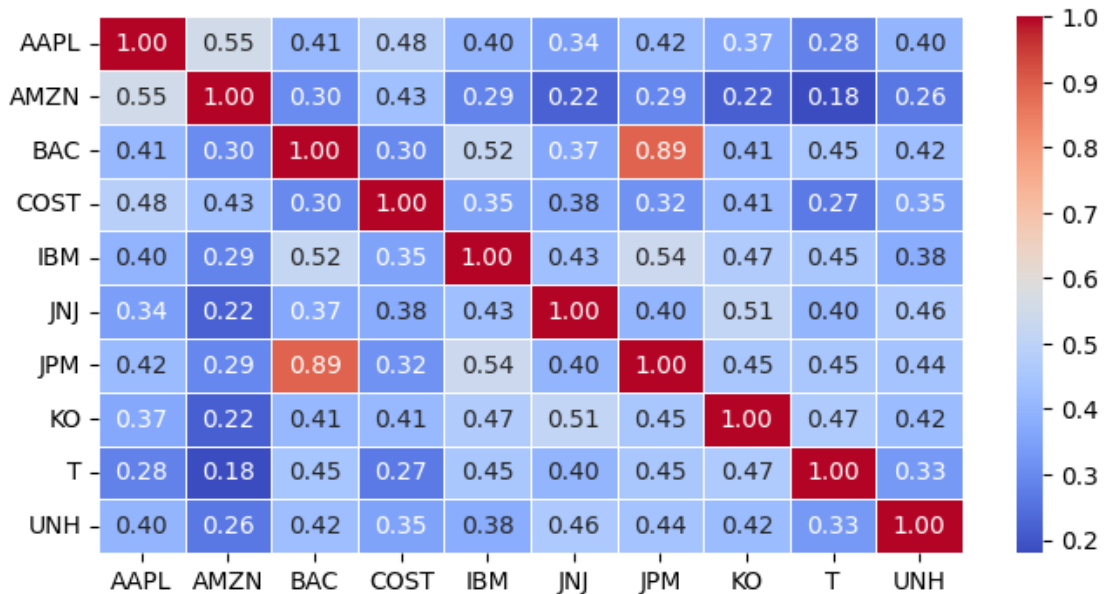
To further analyze the risk, we look at covariance matrix and how does it change over time. We can see that the only two stocks with noticeable positive correlation are Bank of America and JPMorgan Chase, two financial institutions.

In the Figure that shows the evolution of variance over time, we can see the effect of covid19 pandemic much more clearly than in the daily return.

```
[13]: # Calculate the covariance matrix
cov_matrix = stock_returns.cov()
corr_matrix = stock_returns.corr()

# Create a heatmap of the covariance matrix
plt.figure(figsize=(8, 4))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)

plt.title("")
plt.xlabel("")
plt.savefig("correlations.pdf", dpi=300, bbox_inches='tight', transparent=True)
plt.show()
```



```
[14]: rolling_window = 252 # One year

rolling_variance_returns = stock_returns.rolling(window=rolling_window).var()
split_date = pd.to_datetime("2022-12-12")

plt.figure(figsize=(8, 4))

ax = rolling_variance_returns.plot(linewidth=1.3)
ax.axvline(split_date, color='dimgray', linestyle='--', linewidth=1)
ax.axvspan(split_date + pd.Timedelta(days=1), rolling_variance_returns.
           ↪index[-1],
           color='lightgray', alpha=0.3)

ax.set_ylabel("Rolling Variance")
ax.set_xlabel("")
ax.tick_params(axis='x', labelsize=8, rotation=0)
ax.tick_params(axis='y', labelsize=8)

for label in ax.get_xticklabels():
    label.set_horizontalalignment('center')

ax.grid(True)

legend = ax.legend(title="Stock Tickers", loc="upper left", ncol=2)
for line in legend.get_lines():
    line.set_linewidth(2.0)
```

```

y_max = rolling_variance_returns.max().max()
y_text = y_max

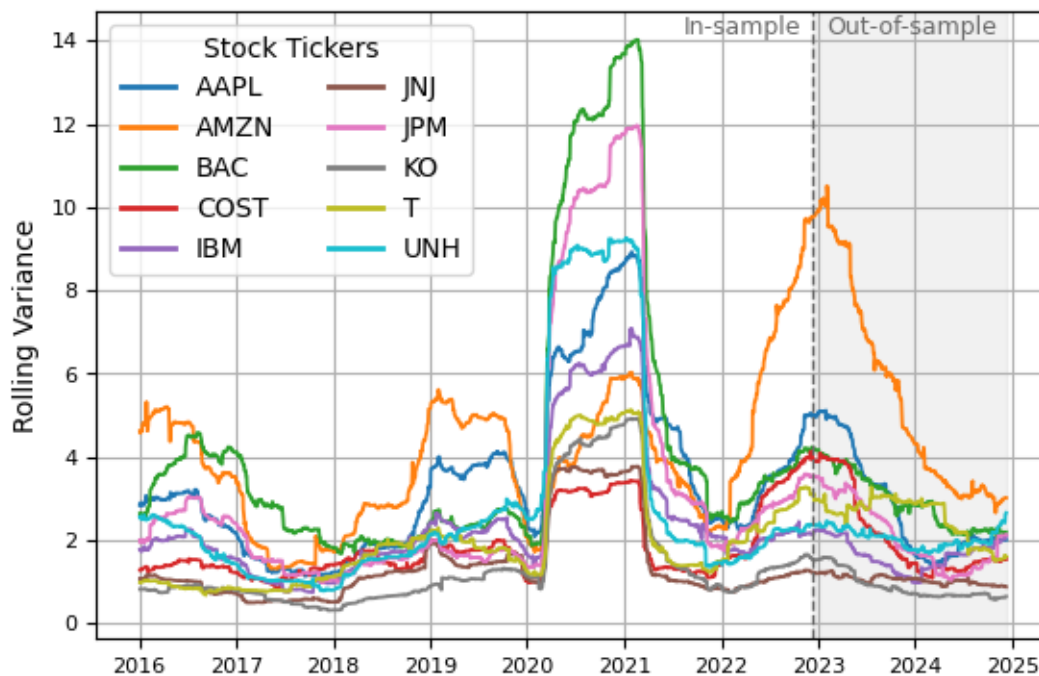
ax.text(split_date - pd.Timedelta(days=40), y_text, "In-sample",
        ha='right', va='bottom', fontsize=9, color='dimgray')

ax.text(split_date + pd.Timedelta(days=50), y_text, "Out-of-sample",
        ha='left', va='bottom', fontsize=9, color='dimgray')

plt.savefig("variances.pdf", dpi=300, bbox_inches='tight', transparent=True)
plt.show()

```

<Figure size 800x400 with 0 Axes>



```

[15]: tickers = list(STOCK_FILES.keys())
      selected_ticker = tickers[7]

      rolling_window = 252

      plt.figure(figsize=(8, 4))

      for ticker in tickers:
          if ticker != selected_ticker:

```

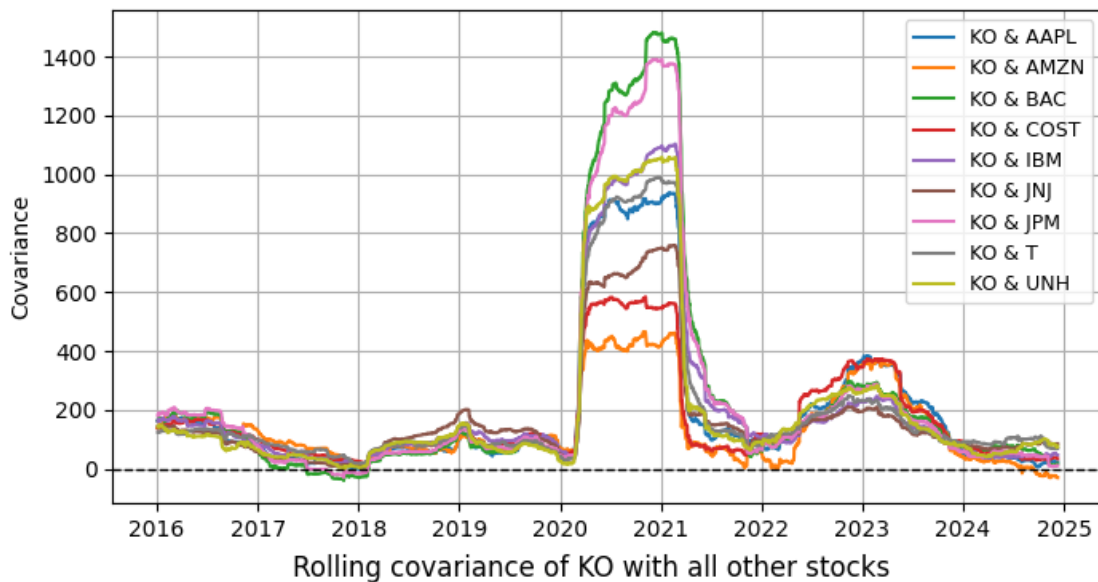
```

    rolling_cov = stock_returns[selected_ticker].
    ↪rolling(window=rolling_window).cov(stock_returns[ticker])*252
    plt.plot(rolling_cov, label=f"{selected_ticker} & {ticker}")

plt.axhline(y=0, color='black', linestyle='--', linewidth=1)
plt.xlabel("")
plt.ylabel("Covariance")
plt.legend(loc="upper right", fontsize=9)
plt.grid(True)
plt.figtext(0.5, -0.005, f"Rolling covariance of {selected_ticker} with all_
    ↪other stocks", ha="center", fontsize=12)

plt.show()

```



Lastly, we will look at autocorrelation of our assets.

```

[16]: max_lag = 3 # Max lag (1, 2, and 3 days)

plt.figure(figsize=(8, 4))

width = 0.1
x_positions = [1, 2, 3]

for i, ticker in enumerate(stock_returns.columns):
    autocorr_values = [stock_returns[ticker].autocorr(lag=lag) for lag in_
    ↪range(1, max_lag + 1)]

```

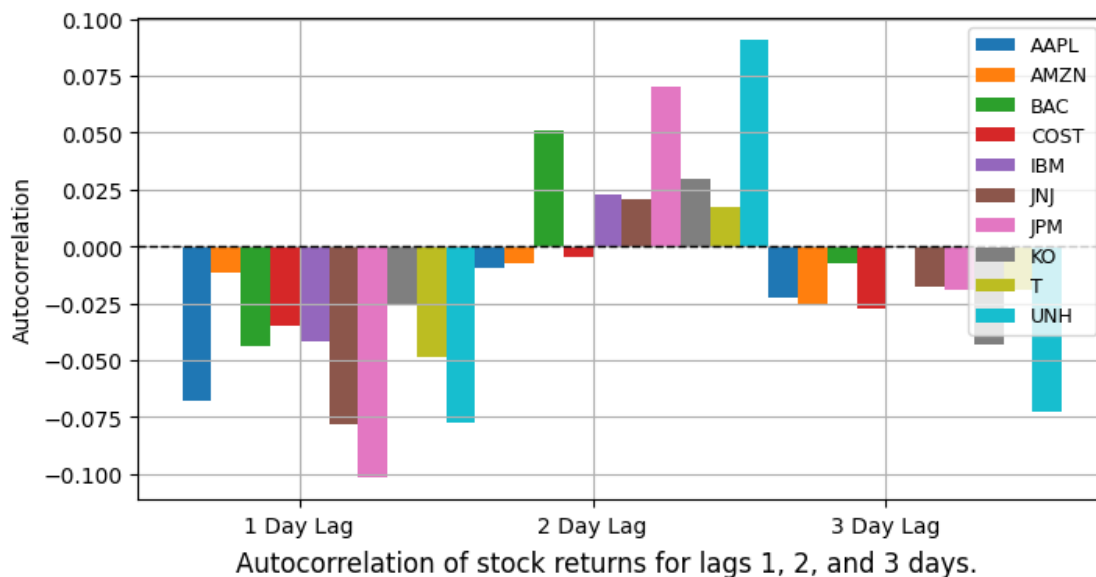
```

plt.bar([x + (i - 3.5) * width for x in x_positions], autocorr_values,
width=width, label=ticker, align='center')

plt.axhline(y=0, color='black', linestyle='--', linewidth=1)
plt.xlabel("")
plt.ylabel("Autocorrelation")
plt.xticks(x_positions, ['1 Day Lag', '2 Day Lag', '3 Day Lag'])
plt.legend(loc="upper right", fontsize=9)
plt.grid(True)
plt.figtext(0.5, -0.005, "Autocorrelation of stock returns for lags 1, 2, and 3_
days.", ha="center", fontsize=12)

plt.show()

```



```

[17]: lag = 1
rolling_window = 252

plt.figure(figsize=(8, 4))

for ticker in stock_returns.columns:
    rolling_autocorr = stock_returns[ticker].rolling(window=rolling_window).
    apply(lambda x: pd.Series(x).autocorr(lag=lag))
    plt.plot(rolling_autocorr, label=ticker, alpha=0.8)

plt.axhline(y=0, color='black', linestyle='--', linewidth=1)
plt.xlabel("")
plt.ylabel("Autocorrelation")

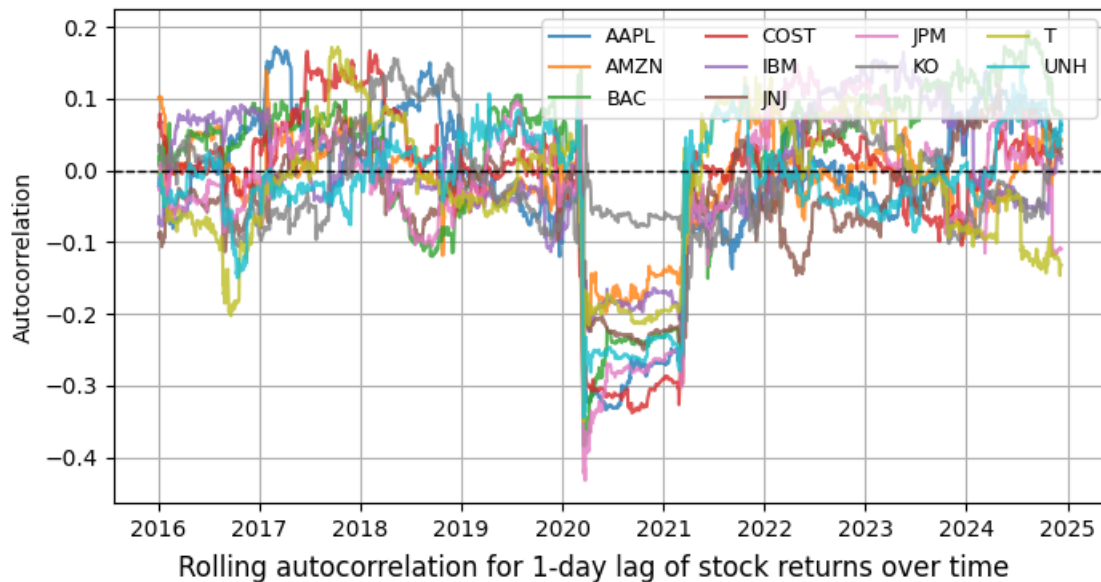
```

```

plt.legend(loc="upper right", fontsize=9, ncol=4)
plt.grid(True)
plt.figtext(0.5, -0.005, f"Rolling autocorrelation for {lag}-day lag of stock_
↳returns over time", ha="center", fontsize=12)

plt.show()

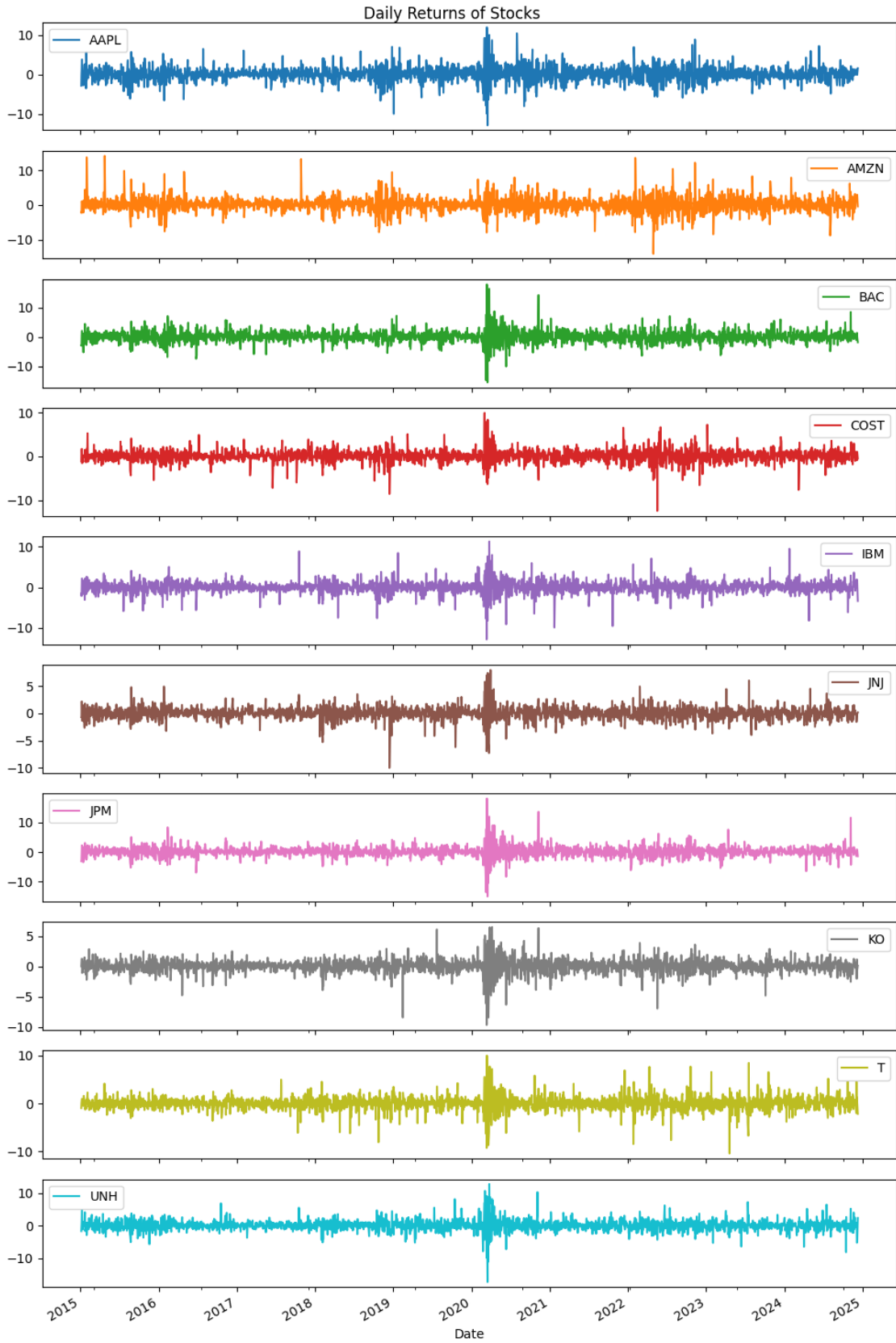
```



```

[18]: # Plotting daily returns
stock_returns.plot(subplots=True, figsize=(10, 15), title='Daily Returns of_
↳Stocks')
plt.tight_layout()
plt.show()

```



```

[19]: #function that produces all necessary information about individual assets to
      ↪ check validity of our data
def explore_data(stock_data):

    print("\n=== Data Exploration ===")

    for ticker, df in stock_data.items():
        print(f"\nAnalysis for {ticker}:")
        print("-" * 50)

        # Basic dataset information
        print("\nDataset Info:")
        print(df.info())

        # Summary statistics
        print("\nSummary Statistics:")
        print(df.describe())

        # Check for missing values
        missing = df.isnull().sum()
        if missing.any():
            print("\nMissing Values:")
            print(missing[missing > 0])

        # Calculate daily returns
        df['Daily_Return'] = df['Close'].pct_change()

        # Basic metrics
        print("\nKey Metrics:")
        print(f"Total Trading Days: {len(df)}")
        print(f"Average Daily Volume: {df['Volume'].mean():.0f}")
        print(f"Average Daily Return: {df['Daily_Return'].mean():.4%}")
        print(f"Return Volatility: {df['Daily_Return'].std():.4%}")
        print(f"Maximum Daily Gain: {df['Daily_Return'].max():.4%}")
        print(f"Maximum Daily Loss: {df['Daily_Return'].min():.4%}")

        # Plot price history
        plt.figure(figsize=(12, 6))
        plt.plot(df.index, df['Close'])
        plt.title(f'{ticker} Stock Price History')
        plt.xlabel('Date')
        plt.ylabel('Price')
        plt.grid(True)
        plt.show()

```

```

# Plot daily returns distribution
plt.figure(figsize=(10, 6))
sns.histplot(df['Daily_Return'], kde=True)
plt.title(f'{ticker} Daily Returns Distribution')
plt.xlabel('Daily Return')
plt.ylabel('Frequency')
plt.show()

```

```
[20]: explore_data(all_data)
```

```
=== Data Exploration ===
```

```
Analysis for AAPL:
```

```
Dataset Info:
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 2501 entries, 0 to 2500
```

```
Data columns (total 10 columns):
```

#	Column	Non-Null Count	Dtype
0	Date	2501 non-null	datetime64[ns]
1	Open	2501 non-null	float64
2	High	2501 non-null	float64
3	Low	2501 non-null	float64
4	Close	2501 non-null	float64
5	Volume	2501 non-null	int64
6	Dividends	2501 non-null	float64
7	Stock Splits	2501 non-null	float64
8	Ticker	2501 non-null	object
9	Daily_Return	2500 non-null	float64

```
dtypes: datetime64[ns](1), float64(7), int64(1), object(1)
```

```
memory usage: 195.5+ KB
```

```
None
```

```
Summary Statistics:
```

	Date	Open	High	Low \
count	2501	2501.000000	2501.000000	2501.000000
mean	2019-12-20 21:48:08.924430336	93.030843	94.022643	92.106531
min	2015-01-02 00:00:00	20.569750	20.951433	20.448615
25%	2017-06-27 00:00:00	35.074528	35.504570	34.754585
50%	2019-12-19 00:00:00	63.756924	64.421819	63.150036
75%	2022-06-14 00:00:00	149.082398	151.550557	147.640055
max	2024-12-09 00:00:00	243.402850	246.645031	241.547325
std	NaN	64.559072	65.227111	63.949739

	Close	Volume	Dividends	Stock Splits	Daily_Return
count	2501.000000	2.501000e+03	2501.000000	2501.000000	2500.000000
mean	93.113738	1.174991e+08	0.003066	0.001599	0.108813
min	20.647455	2.404830e+07	0.000000	0.000000	-12.864693
25%	35.069870	7.137960e+07	0.000000	0.000000	-0.734672
50%	63.901985	1.005820e+08	0.000000	0.000000	0.097043
75%	149.472794	1.426898e+08	0.000000	0.000000	1.014775
max	246.156204	6.488252e+08	0.250000	4.000000	11.980810
std	64.631061	6.836079e+07	0.024585	0.079984	1.796391

Missing Values:

Daily_Return 1

dtype: int64

Key Metrics:

Total Trading Days: 2501

Average Daily Volume: 117,499,087

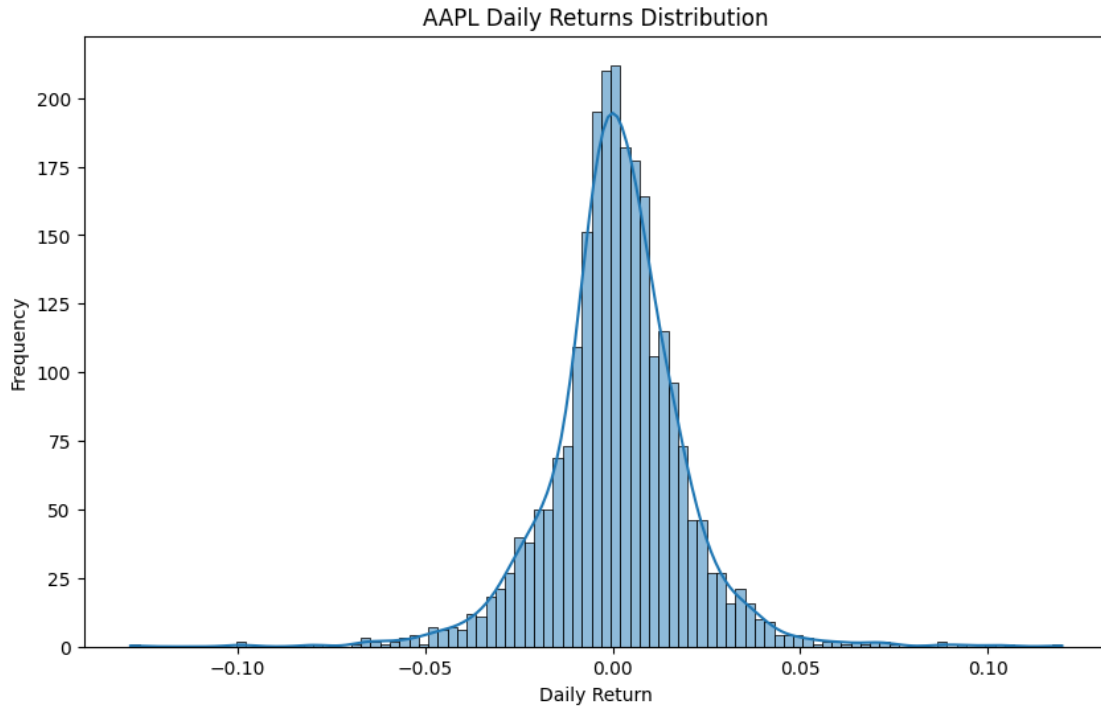
Average Daily Return: 0.1088%

Return Volatility: 1.7964%

Maximum Daily Gain: 11.9808%

Maximum Daily Loss: -12.8647%





Analysis for AMZN:

Dataset Info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2501 entries, 0 to 2500

Data columns (total 10 columns):

#	Column	Non-Null Count	Dtype
0	Date	2501 non-null	datetime64[ns]
1	Open	2501 non-null	float64
2	High	2501 non-null	float64
3	Low	2501 non-null	float64
4	Close	2501 non-null	float64
5	Volume	2501 non-null	int64
6	Dividends	2501 non-null	float64
7	Stock Splits	2501 non-null	float64
8	Ticker	2501 non-null	object
9	Daily_Return	2500 non-null	float64

dtypes: datetime64[ns](1), float64(7), int64(1), object(1)

memory usage: 195.5+ KB

None

Summary Statistics:

	Date	Open	High	Low	\
count	2501	2501.000000	2501.000000	2501.000000	
mean	2019-12-20 21:48:08.924430336	100.505960	101.649652	99.262112	
min	2015-01-02 00:00:00	14.314000	14.539500	14.262500	
25%	2017-06-27 00:00:00	48.450001	48.740501	48.015999	
50%	2019-12-19 00:00:00	94.290001	95.347000	93.207497	
75%	2022-06-14 00:00:00	153.529999	155.076996	151.029999	
max	2024-12-09 00:00:00	227.210007	230.080002	225.669998	
std	NaN	53.484189	54.103002	52.818208	

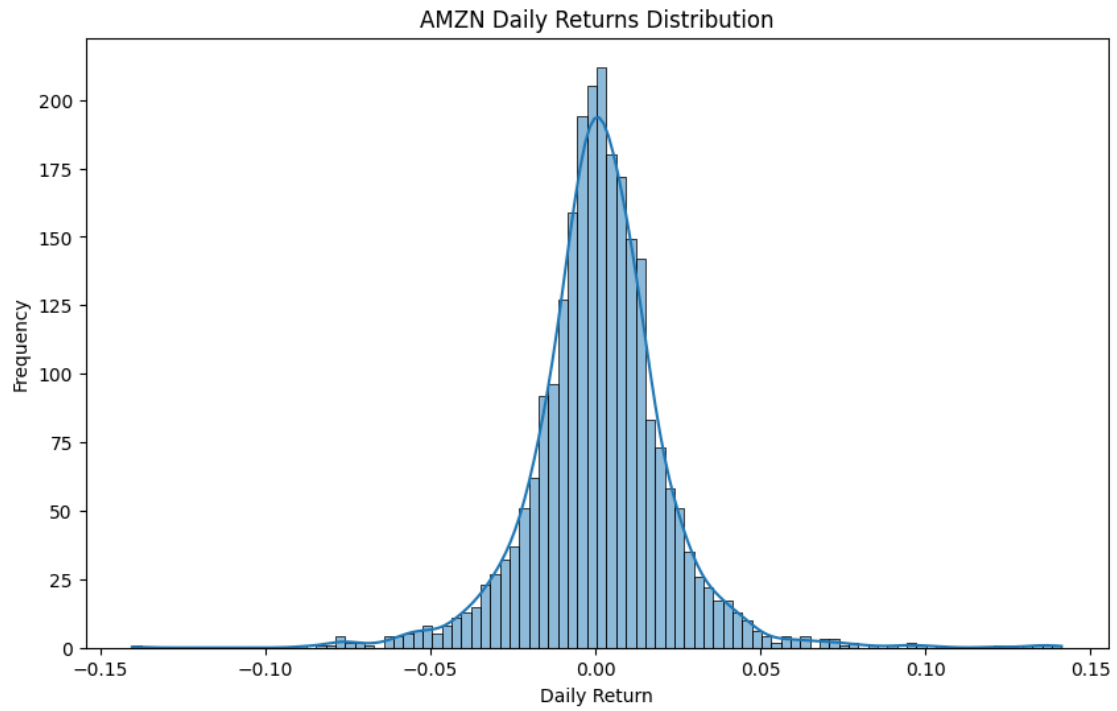
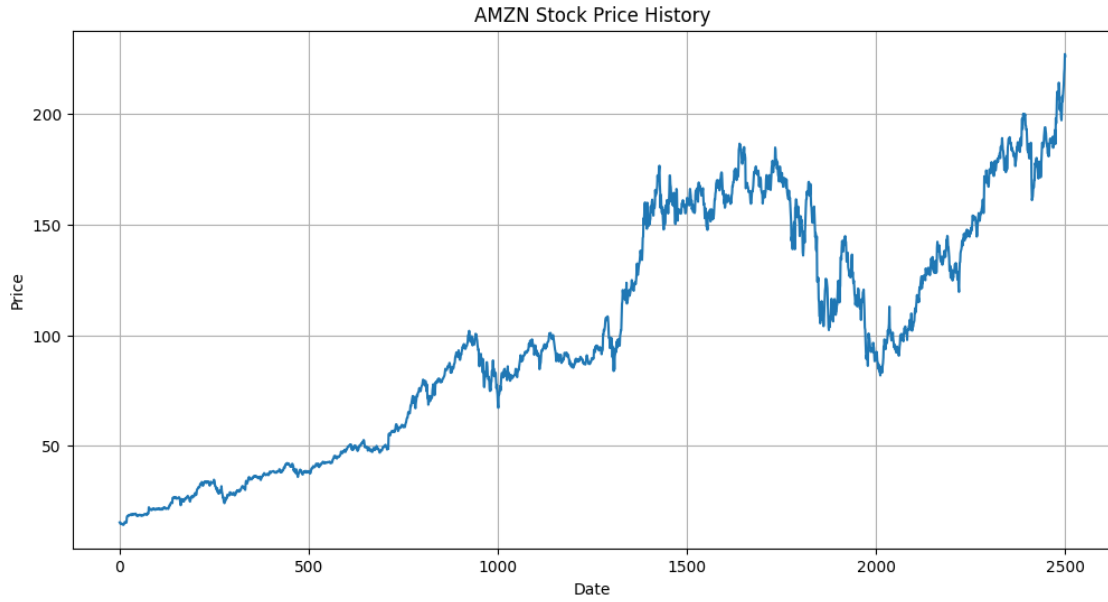
	Close	Volume	Dividends	Stock Splits	Daily_Return
count	2501.000000	2.501000e+03	2501.0	2501.000000	2500.000000
mean	100.484731	7.639866e+07	0.0	0.007997	0.128696
min	14.347500	1.762600e+07	0.0	0.000000	-14.049438
25%	48.314999	5.094190e+07	0.0	0.000000	-0.861898
50%	94.315002	6.538400e+07	0.0	0.000000	0.116871
75%	153.339996	9.030000e+07	0.0	0.000000	1.130712
max	227.029999	4.771220e+08	0.0	20.000000	14.131126
std	53.463575	4.058874e+07	0.0	0.399920	2.066849

Missing Values:

Daily_Return 1
dtype: int64

Key Metrics:

Total Trading Days: 2501
Average Daily Volume: 76,398,658
Average Daily Return: 0.1287%
Return Volatility: 2.0668%
Maximum Daily Gain: 14.1311%
Maximum Daily Loss: -14.0494%



Analysis for BAC:

Dataset Info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2501 entries, 0 to 2500

Data columns (total 10 columns):

#	Column	Non-Null Count	Dtype
0	Date	2501 non-null	datetime64[ns]
1	Open	2501 non-null	float64
2	High	2501 non-null	float64
3	Low	2501 non-null	float64
4	Close	2501 non-null	float64
5	Volume	2501 non-null	int64
6	Dividends	2501 non-null	float64
7	Stock Splits	2501 non-null	float64
8	Ticker	2501 non-null	object
9	Daily_Return	2500 non-null	float64

dtypes: datetime64[ns](1), float64(7), int64(1), object(1)

memory usage: 195.5+ KB

None

Summary Statistics:

	Date	Open	High	Low
count	2501	2501.000000	2501.000000	2501.000000
mean	2019-12-20 21:48:08.924430336	25.669178	25.946988	25.390863
min	2015-01-02 00:00:00	9.303897	9.376964	8.922323
25%	2017-06-27 00:00:00	19.591599	19.890203	19.392323
50%	2019-12-19 00:00:00	25.437735	25.700158	25.205592
75%	2022-06-14 00:00:00	31.994226	32.352552	31.643129
max	2024-12-09 00:00:00	46.980381	47.235819	46.813367
std	NaN	8.869010	8.957676	8.784658

	Close	Volume	Dividends	Stock Splits	Daily_Return
count	2501.000000	2.501000e+03	2501.000000	2501.0	2500.000000
mean	25.669285	6.451388e+07	0.002527	0.0	0.065259
min	9.060339	1.380540e+07	0.000000	0.0	-15.397343
25%	19.641262	4.185550e+07	0.000000	0.0	-0.917979
50%	25.412992	5.534250e+07	0.000000	0.0	0.037706
75%	32.039276	7.736920e+07	0.000000	0.0	1.038737
max	46.931259	3.750887e+08	0.260000	0.0	17.796180
std	8.868818	3.454576e+07	0.021577	0.0	1.964254

Missing Values:

Daily_Return 1

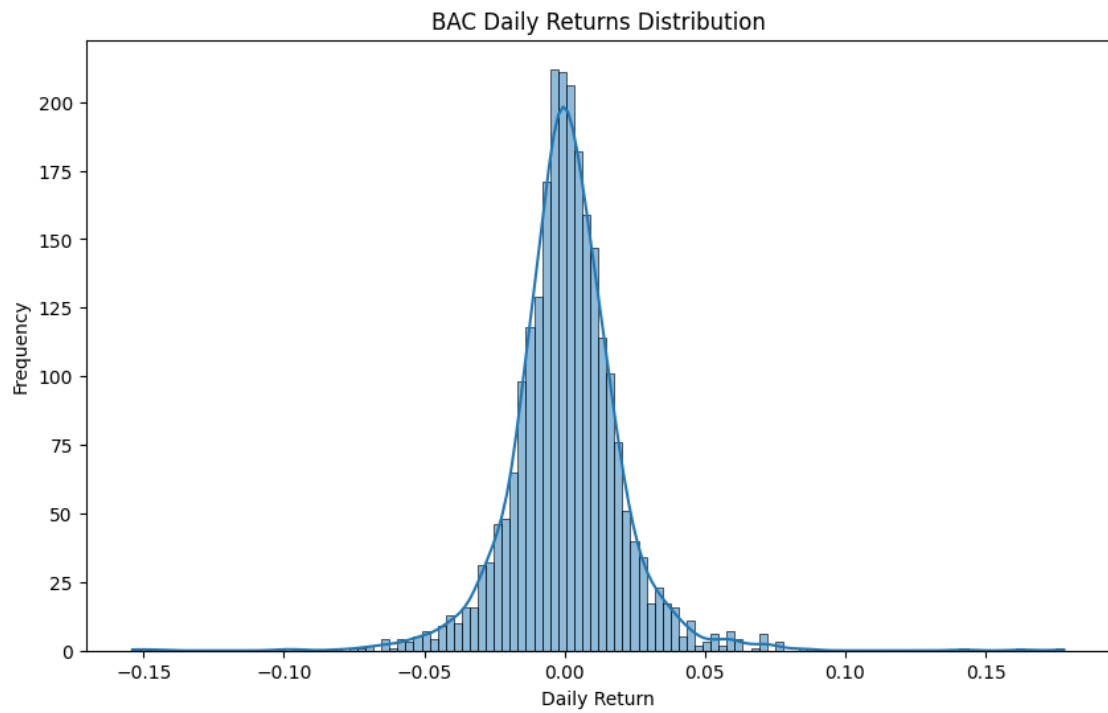
dtype: int64

Key Metrics:

Total Trading Days: 2501

Average Daily Volume: 64,513,878

Average Daily Return: 0.0653%
Return Volatility: 1.9643%
Maximum Daily Gain: 17.7962%
Maximum Daily Loss: -15.3973%



Analysis for COST:

Dataset Info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2501 entries, 0 to 2500

Data columns (total 10 columns):

#	Column	Non-Null Count	Dtype
0	Date	2501 non-null	datetime64[ns]
1	Open	2501 non-null	float64
2	High	2501 non-null	float64
3	Low	2501 non-null	float64
4	Close	2501 non-null	float64
5	Volume	2501 non-null	int64
6	Dividends	2501 non-null	float64
7	Stock Splits	2501 non-null	float64
8	Ticker	2501 non-null	object
9	Daily_Return	2500 non-null	float64

dtypes: datetime64[ns](1), float64(7), int64(1), object(1)

memory usage: 195.5+ KB

None

Summary Statistics:

	Date	Open	High	Low
count	2501	2501.000000	2501.000000	2501.000000
mean	2019-12-20 21:48:08.924430336	332.289000	335.142735	329.430257
min	2015-01-02 00:00:00	111.069483	112.912066	98.495577
25%	2017-06-27 00:00:00	144.190941	145.722821	143.662390
50%	2019-12-19 00:00:00	272.684612	274.762174	270.583624
75%	2022-06-14 00:00:00	477.084797	480.970229	472.582263
max	2024-12-09 00:00:00	994.613689	995.312022	979.899144
std	NaN	211.406767	213.240316	209.494859

	Close	Volume	Dividends	Stock Splits	Daily_Return
count	2501.000000	2.501000e+03	2501.000000	2501.0	2500.000000
mean	332.478326	2.324952e+06	0.025892	0.0	0.095437
min	111.692291	4.910000e+05	0.000000	0.0	-12.451340
25%	144.491669	1.609100e+06	0.000000	0.0	-0.550858
50%	272.653564	1.994400e+06	0.000000	0.0	0.127245
75%	476.920929	2.598500e+06	0.000000	0.0	0.770821
max	990.224243	2.423300e+07	15.000000	0.0	9.959463
std	211.540465	1.292744e+06	0.409284	0.0	1.360922

Missing Values:

Daily_Return 1

dtype: int64

Key Metrics:

Total Trading Days: 2501

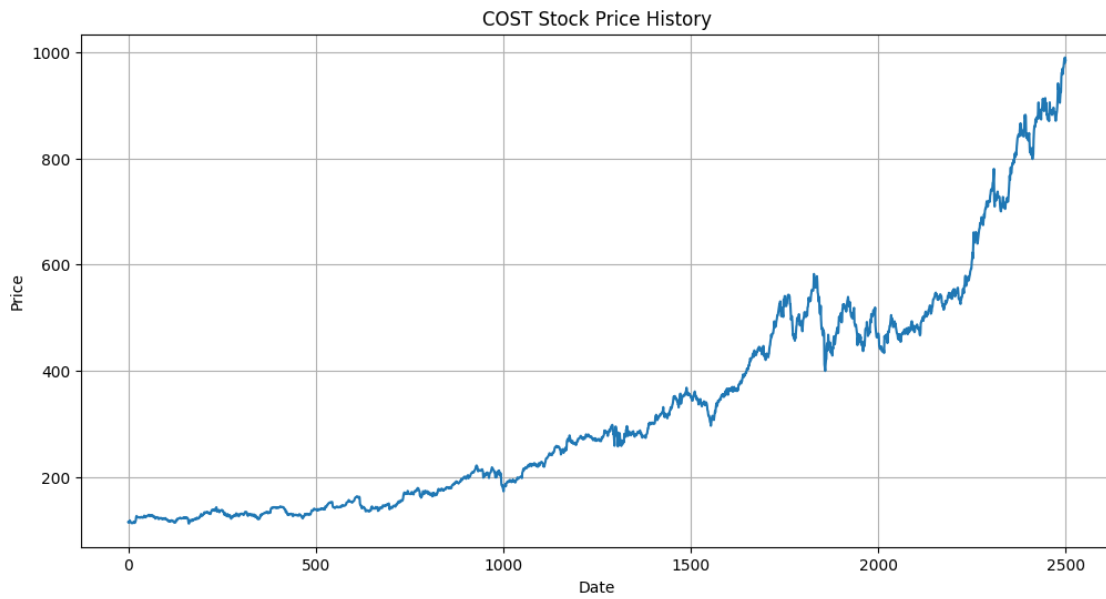
Average Daily Volume: 2,324,952

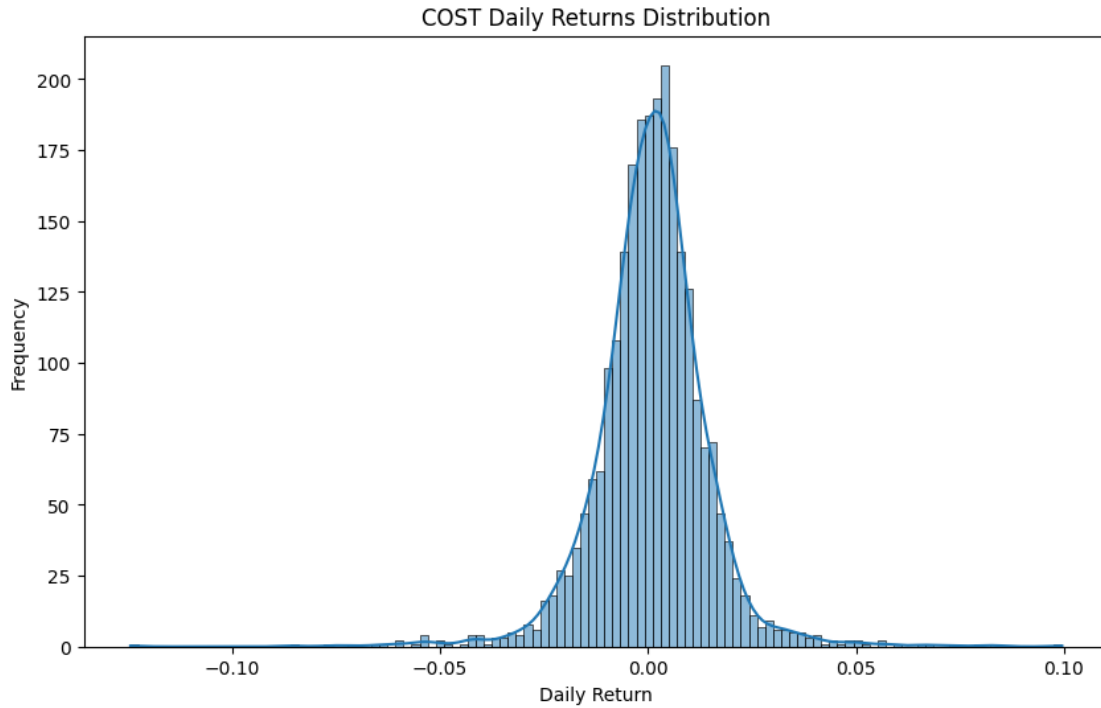
Average Daily Return: 0.0954%

Return Volatility: 1.3609%

Maximum Daily Gain: 9.9595%

Maximum Daily Loss: -12.4513%





Analysis for IBM:

Dataset Info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2501 entries, 0 to 2500

Data columns (total 10 columns):

#	Column	Non-Null Count	Dtype
0	Date	2501 non-null	datetime64[ns]
1	Open	2501 non-null	float64
2	High	2501 non-null	float64
3	Low	2501 non-null	float64
4	Close	2501 non-null	float64
5	Volume	2501 non-null	int64
6	Dividends	2501 non-null	float64
7	Stock Splits	2501 non-null	float64
8	Ticker	2501 non-null	object
9	Daily_Return	2500 non-null	float64

dtypes: datetime64[ns](1), float64(7), int64(1), object(1)

memory usage: 195.5+ KB

None

Summary Statistics:

	Date	Open	High	Low	\
count	2501	2501.000000	2501.000000	2501.000000	
mean	2019-12-20 21:48:08.924430336	113.396382	114.320986	112.487754	
min	2015-01-02 00:00:00	71.465075	73.837167	68.413076	
25%	2017-06-27 00:00:00	98.226438	98.890701	97.498198	
50%	2019-12-19 00:00:00	104.326784	105.037205	103.664300	
75%	2022-06-14 00:00:00	117.144356	118.188562	116.248079	
max	2024-12-09 00:00:00	234.862005	236.194211	231.131848	
std	NaN	27.537544	27.761785	27.349045	

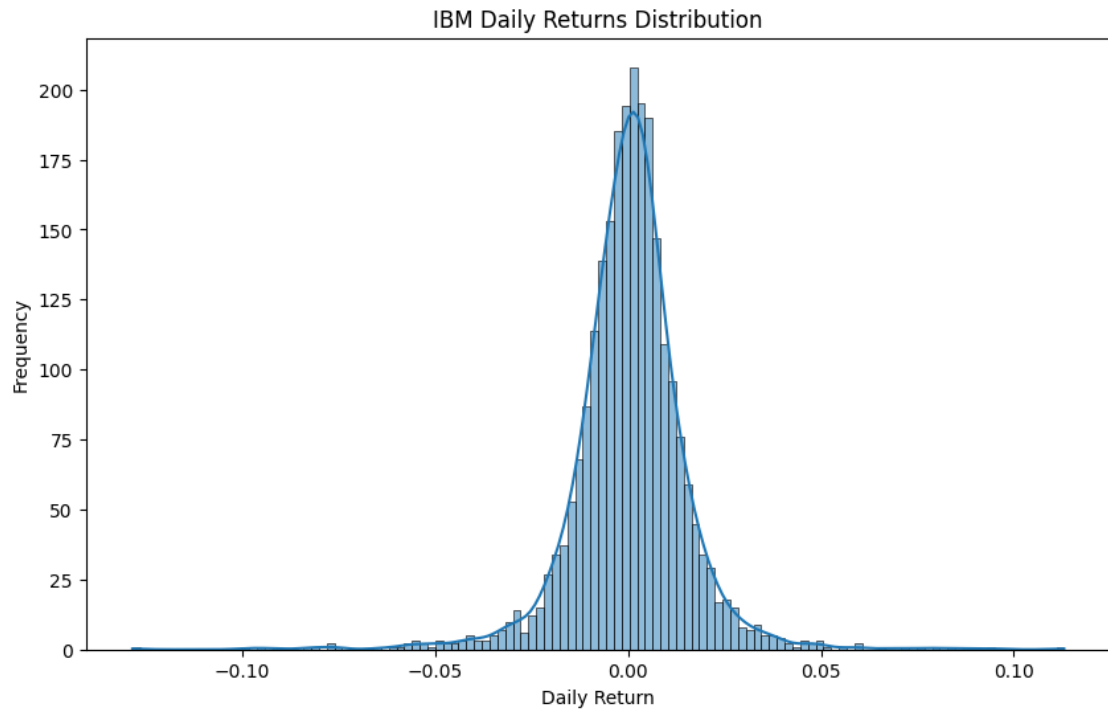
	Close	Volume	Dividends	Stock Splits	Daily_Return
count	2501.000000	2.501000e+03	2501.000000	2501.000000	2500.000000
mean	113.430841	4.801913e+06	0.024075	0.000418	0.044387
min	71.593498	1.247878e+06	0.000000	0.000000	-12.850751
25%	98.214211	3.285300e+06	0.000000	0.000000	-0.628358
50%	104.313705	4.105341e+06	0.000000	0.000000	0.077322
75%	117.263985	5.349453e+06	0.000000	0.000000	0.740207
max	234.901474	3.981442e+07	1.670000	1.046000	11.301055
std	27.611577	2.866284e+06	0.189864	0.020916	1.501577

Missing Values:

Daily_Return 1
dtype: int64

Key Metrics:

Total Trading Days: 2501
Average Daily Volume: 4,801,913
Average Daily Return: 0.0444%
Return Volatility: 1.5016%
Maximum Daily Gain: 11.3011%
Maximum Daily Loss: -12.8508%



Analysis for JNJ:

Dataset Info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2501 entries, 0 to 2500

Data columns (total 10 columns):

#	Column	Non-Null Count	Dtype
0	Date	2501 non-null	datetime64[ns]
1	Open	2501 non-null	float64
2	High	2501 non-null	float64
3	Low	2501 non-null	float64
4	Close	2501 non-null	float64
5	Volume	2501 non-null	int64
6	Dividends	2501 non-null	float64
7	Stock Splits	2501 non-null	float64
8	Ticker	2501 non-null	object
9	Daily_Return	2500 non-null	float64

dtypes: datetime64[ns](1), float64(7), int64(1), object(1)

memory usage: 195.5+ KB

None

Summary Statistics:

	Date	Open	High	Low
count	2501	2501.000000	2501.000000	2501.000000
mean	2019-12-20 21:48:08.924430336	122.184105	123.047551	121.282361
min	2015-01-02 00:00:00	69.193810	70.094033	62.396503
25%	2017-06-27 00:00:00	101.063175	101.715434	100.525612
50%	2019-12-19 00:00:00	120.199698	121.224852	118.887017
75%	2022-06-14 00:00:00	148.796474	149.890001	147.740559
max	2024-12-09 00:00:00	167.970638	169.413512	167.135763
std	NaN	27.797750	28.012420	27.581009

	Close	Volume	Dividends	Stock Splits	Daily_Return
count	2501.000000	2.501000e+03	2501.000000	2501.0	2500.000000
mean	122.193028	7.716026e+06	0.015570	0.0	0.031869
min	69.216728	2.114900e+06	0.000000	0.0	-10.037886
25%	100.999519	5.556600e+06	0.000000	0.0	-0.494468
50%	120.233482	6.713600e+06	0.000000	0.0	0.034192
75%	148.855362	8.374400e+06	0.000000	0.0	0.592710
max	168.796432	1.513195e+08	1.240000	0.0	7.997706
std	27.798175	6.016613e+06	0.123816	0.0	1.137935

Missing Values:

Daily_Return 1

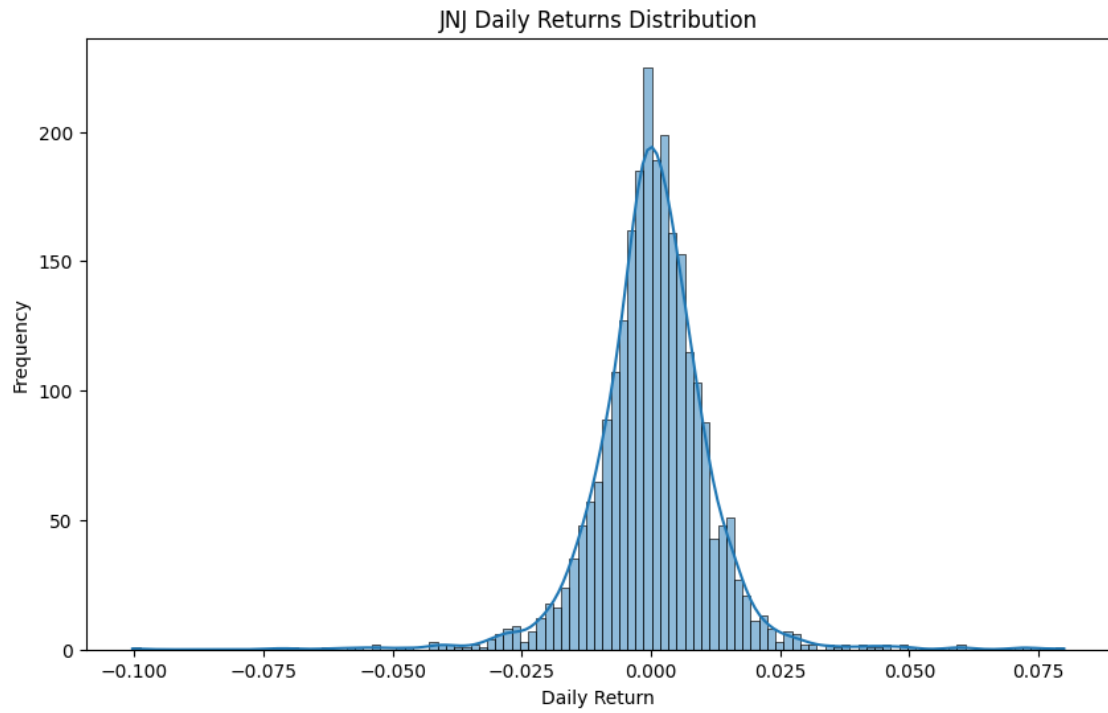
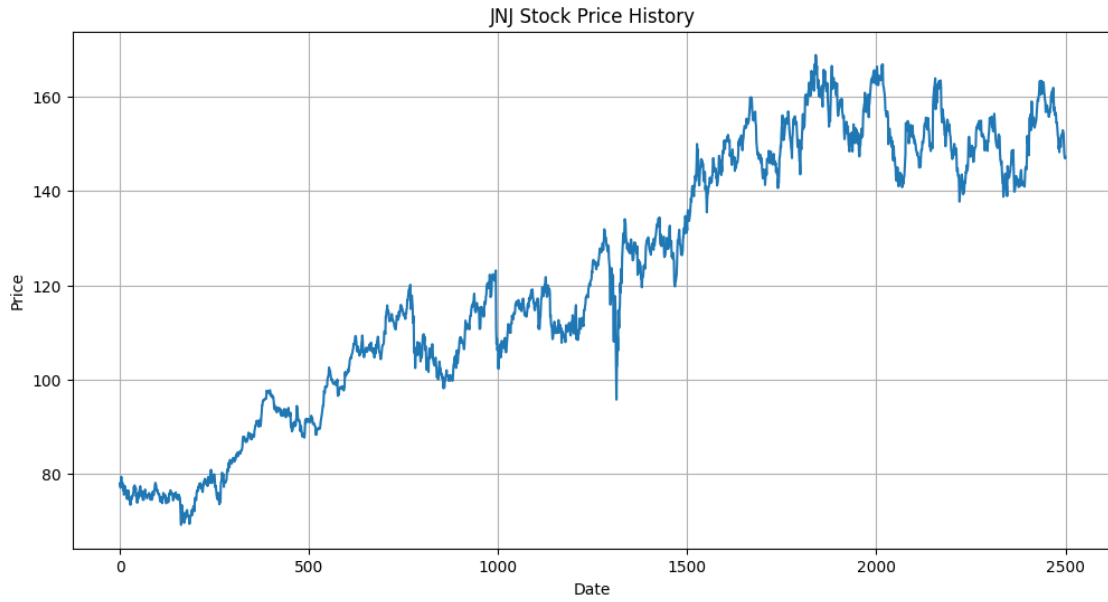
dtype: int64

Key Metrics:

Total Trading Days: 2501

Average Daily Volume: 7,716,026

Average Daily Return: 0.0319%
Return Volatility: 1.1379%
Maximum Daily Gain: 7.9977%
Maximum Daily Loss: -10.0379%



Analysis for JPM:

Dataset Info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2501 entries, 0 to 2500

Data columns (total 10 columns):

#	Column	Non-Null Count	Dtype
0	Date	2501 non-null	datetime64[ns]
1	Open	2501 non-null	float64
2	High	2501 non-null	float64
3	Low	2501 non-null	float64
4	Close	2501 non-null	float64
5	Volume	2501 non-null	int64
6	Dividends	2501 non-null	float64
7	Stock Splits	2501 non-null	float64
8	Ticker	2501 non-null	object
9	Daily_Return	2500 non-null	float64

dtypes: datetime64[ns](1), float64(7), int64(1), object(1)

memory usage: 195.5+ KB

None

Summary Statistics:

	Date	Open	High	Low
count	2501	2501.000000	2501.000000	2501.000000
mean	2019-12-20 21:48:08.924430336	103.899354	104.868975	102.943867
min	2015-01-02 00:00:00	40.967809	41.611079	38.114600
25%	2017-06-27 00:00:00	72.486878	73.128980	71.797543
50%	2019-12-19 00:00:00	93.851143	94.553588	93.186882
75%	2022-06-14 00:00:00	133.334972	134.808876	132.108251
max	2024-12-09 00:00:00	246.489895	250.248868	245.466485
std	NaN	43.870729	44.252062	43.519625

	Close	Volume	Dividends	Stock Splits	Daily_Return
count	2501.000000	2.501000e+03	2501.000000	2501.0	2500.000000
mean	103.923547	1.397199e+07	0.012511	0.0	0.080131
min	40.855129	3.220500e+06	0.000000	0.0	-14.964867
25%	72.448341	9.666100e+06	0.000000	0.0	-0.713477
50%	93.790161	1.260320e+07	0.000000	0.0	0.059864
75%	133.647903	1.628000e+07	0.000000	0.0	0.874642
max	246.293060	5.619230e+07	1.250000	0.0	18.012479
std	43.901077	6.545985e+06	0.103152	0.0	1.726695

Missing Values:

Daily_Return 1

dtype: int64

Key Metrics:

Total Trading Days: 2501

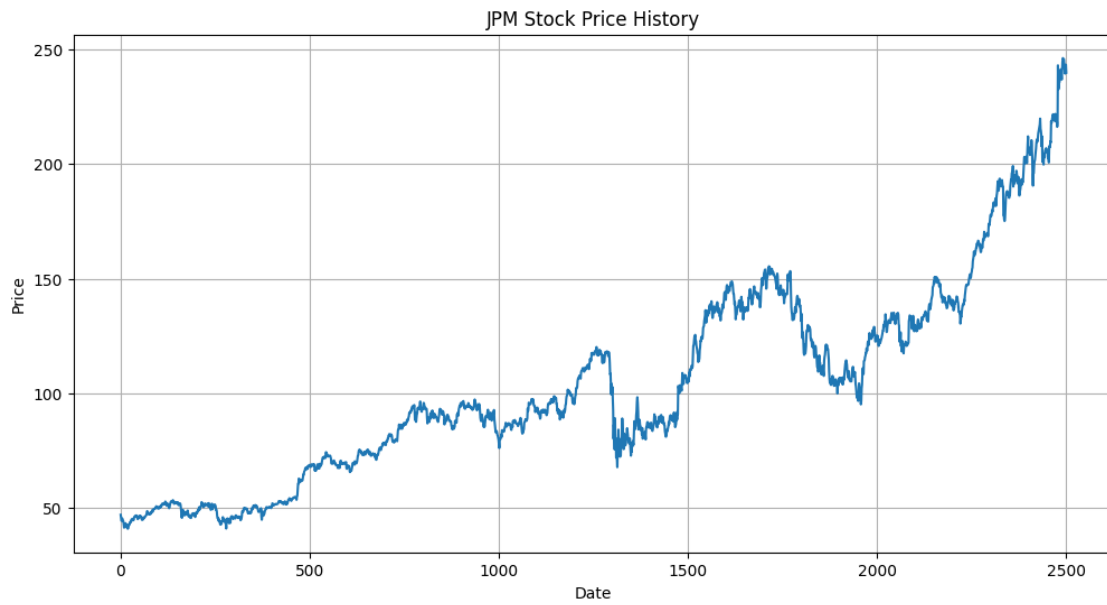
Average Daily Volume: 13,971,993

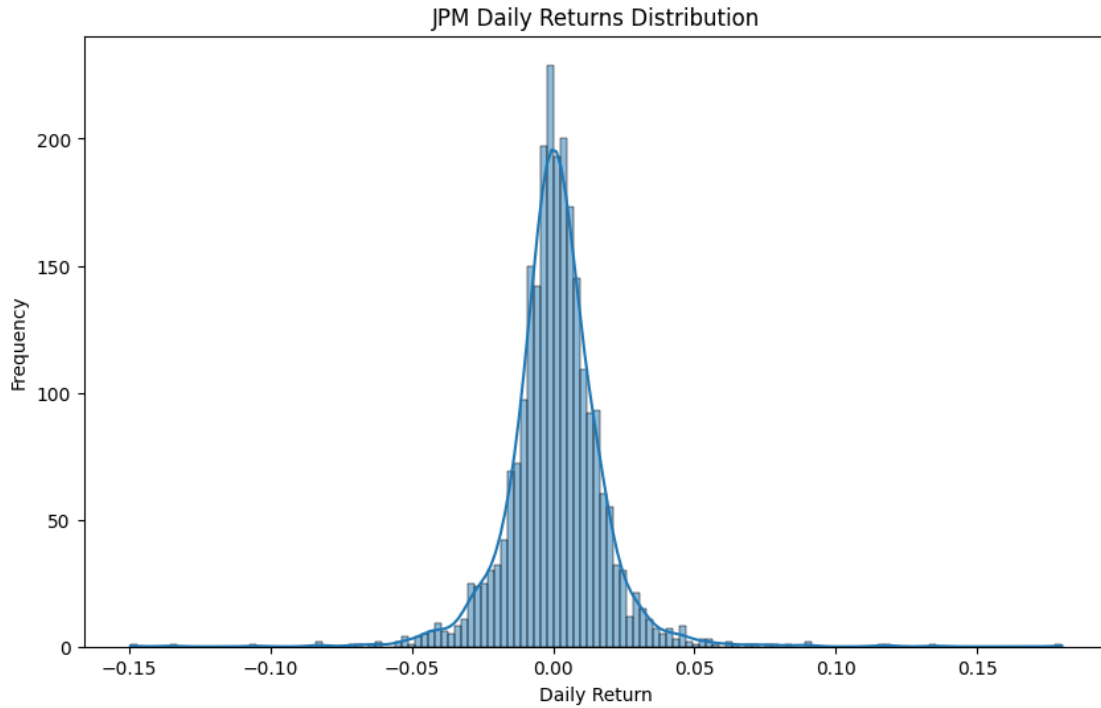
Average Daily Return: 0.0801%

Return Volatility: 1.7267%

Maximum Daily Gain: 18.0125%

Maximum Daily Loss: -14.9649%





Analysis for KO:

Dataset Info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2501 entries, 0 to 2500

Data columns (total 10 columns):

#	Column	Non-Null Count	Dtype
0	Date	2501 non-null	datetime64[ns]
1	Open	2501 non-null	float64
2	High	2501 non-null	float64
3	Low	2501 non-null	float64
4	Close	2501 non-null	float64
5	Volume	2501 non-null	int64
6	Dividends	2501 non-null	float64
7	Stock Splits	2501 non-null	float64
8	Ticker	2501 non-null	object
9	Daily_Return	2500 non-null	float64

dtypes: datetime64[ns](1), float64(7), int64(1), object(1)

memory usage: 195.5+ KB

None

Summary Statistics:

	Date	Open	High	Low	\
count	2501	2501.000000	2501.000000	2501.000000	
mean	2019-12-20 21:48:08.924430336	44.001065	44.295100	43.693347	
min	2015-01-02 00:00:00	27.698997	28.086046	26.656366	
25%	2017-06-27 00:00:00	34.419227	34.655227	34.197051	
50%	2019-12-19 00:00:00	42.481516	42.821463	42.115230	
75%	2022-06-14 00:00:00	54.880057	55.309230	54.438788	
max	2024-12-09 00:00:00	71.217255	71.440715	70.197086	
std	NaN	10.826422	10.901142	10.741245	

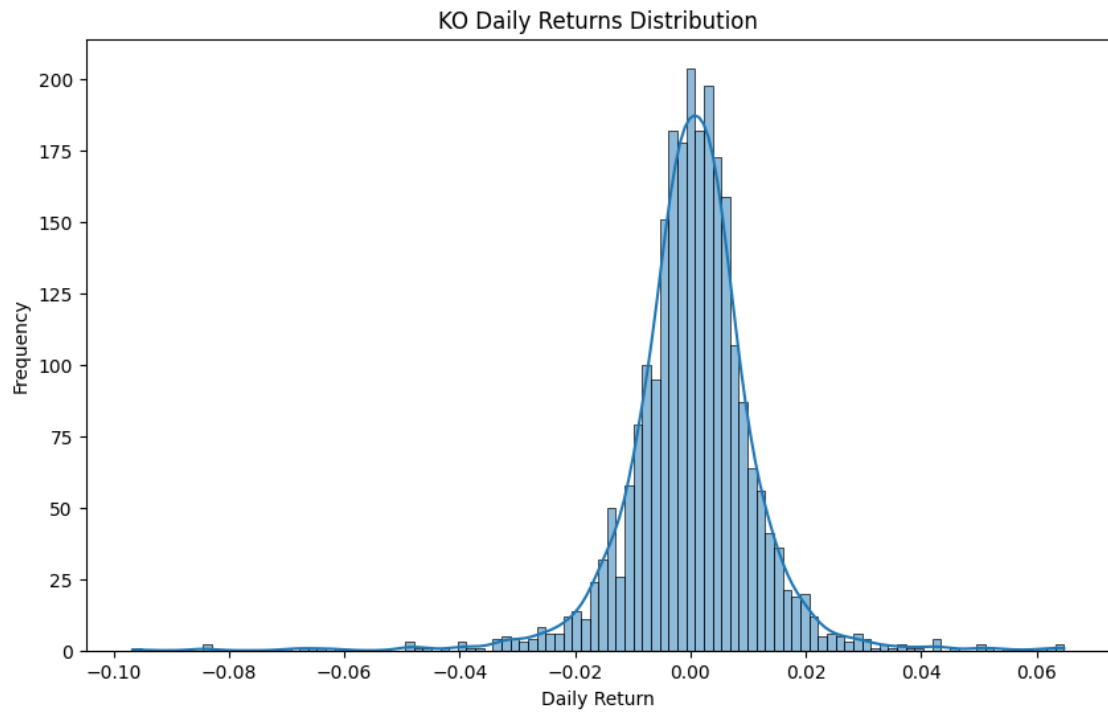
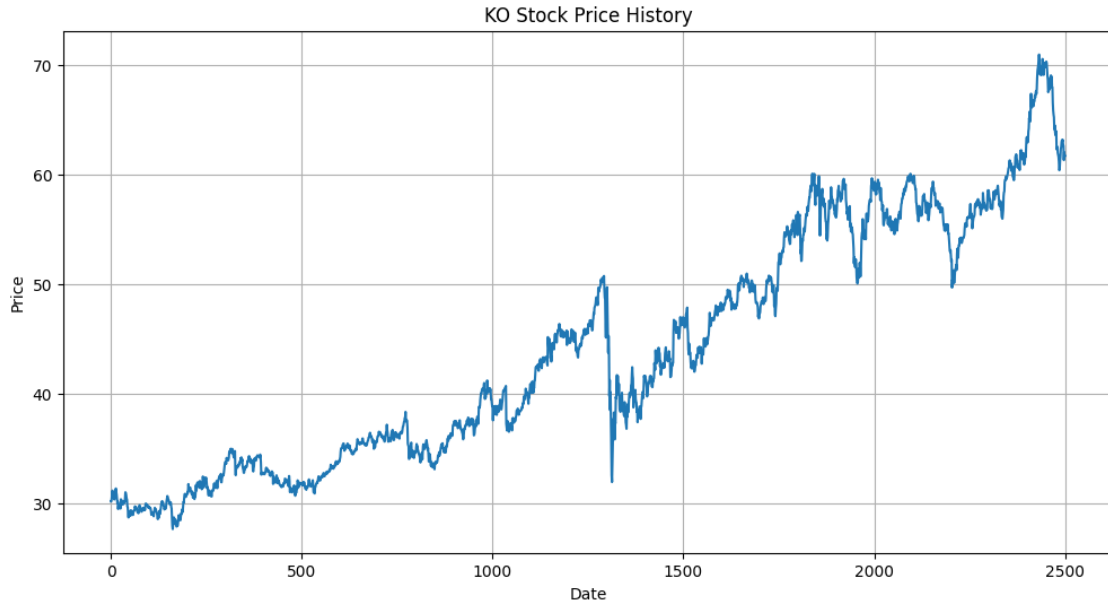
	Close	Volume	Dividends	Stock Splits	Daily_Return
count	2501.000000	2.501000e+03	2501.000000	2501.0	2500.000000
mean	43.998981	1.409036e+07	0.006485	0.0	0.034881
min	27.698984	2.996300e+06	0.000000	0.0	-9.672484
25%	34.403275	1.033610e+07	0.000000	0.0	-0.458286
50%	42.423729	1.281890e+07	0.000000	0.0	0.063394
75%	54.844040	1.597560e+07	0.000000	0.0	0.575869
max	70.935493	6.784570e+07	0.485000	0.0	6.479564
std	10.827818	6.104020e+06	0.051210	0.0	1.121857

Missing Values:

Daily_Return 1
dtype: int64

Key Metrics:

Total Trading Days: 2501
Average Daily Volume: 14,090,359
Average Daily Return: 0.0349%
Return Volatility: 1.1219%
Maximum Daily Gain: 6.4796%
Maximum Daily Loss: -9.6725%



Analysis for T:

Dataset Info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2501 entries, 0 to 2500

Data columns (total 10 columns):

#	Column	Non-Null Count	Dtype
0	Date	2501 non-null	datetime64[ns]
1	Open	2501 non-null	float64
2	High	2501 non-null	float64
3	Low	2501 non-null	float64
4	Close	2501 non-null	float64
5	Volume	2501 non-null	int64
6	Dividends	2501 non-null	float64
7	Stock Splits	2501 non-null	float64
8	Ticker	2501 non-null	object
9	Daily_Return	2500 non-null	float64

dtypes: datetime64[ns](1), float64(7), int64(1), object(1)

memory usage: 195.5+ KB

None

Summary Statistics:

	Date	Open	High	Low
count	2501	2501.000000	2501.000000	2501.000000
mean	2019-12-20 21:48:08.924430336	15.558189	15.686849	15.429546
min	2015-01-02 00:00:00	11.562619	11.658325	11.385775
25%	2017-06-27 00:00:00	14.437178	14.563165	14.312028
50%	2019-12-19 00:00:00	15.636639	15.760540	15.494074
75%	2022-06-14 00:00:00	16.485026	16.640096	16.366114
max	2024-12-09 00:00:00	23.121545	23.247314	22.811970
std	NaN	1.939097	1.958834	1.927265

	Close	Volume	Dividends	Stock Splits	Daily_Return
count	2501.000000	2.501000e+03	2501.000000	2501.000000	2500.000000
mean	15.559822	4.261030e+07	0.007012	0.000529	0.036349
min	11.562619	9.085818e+06	0.000000	0.000000	-10.406097
25%	14.427429	2.899600e+07	0.000000	0.000000	-0.626029
50%	15.624257	3.732850e+07	0.000000	0.000000	0.059353
75%	16.498415	4.958923e+07	0.000000	0.000000	0.709808
max	23.102198	3.270974e+08	0.520000	1.324000	10.022340
std	1.947871	2.227826e+07	0.056451	0.026475	1.431952

Missing Values:

Daily_Return 1

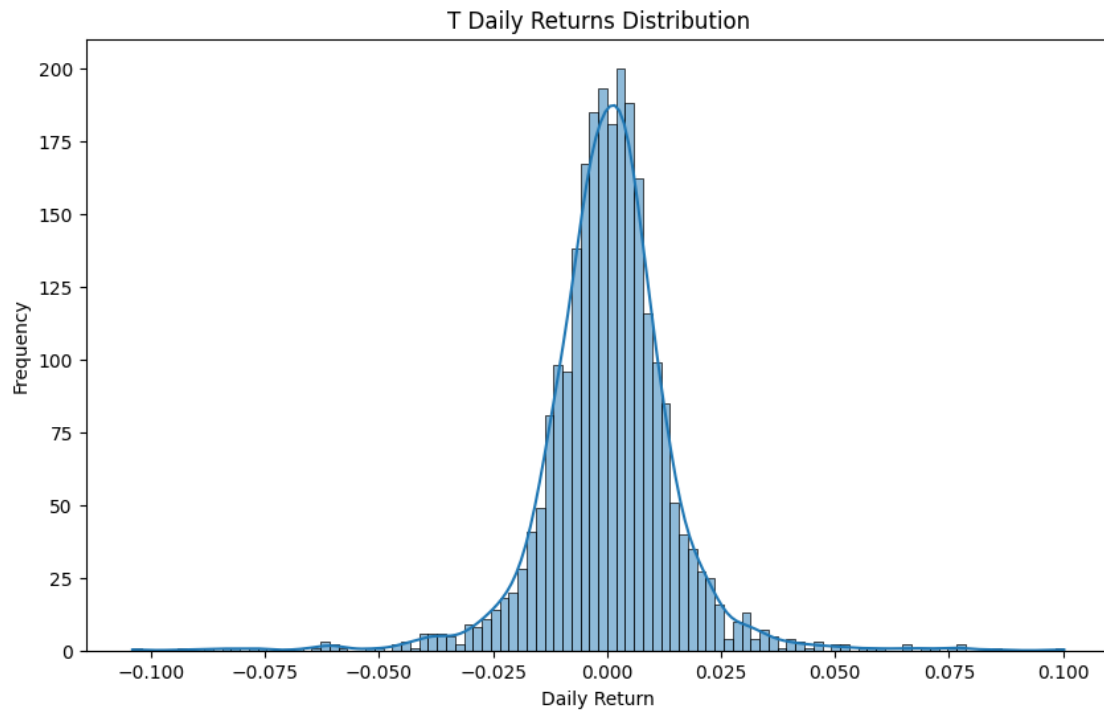
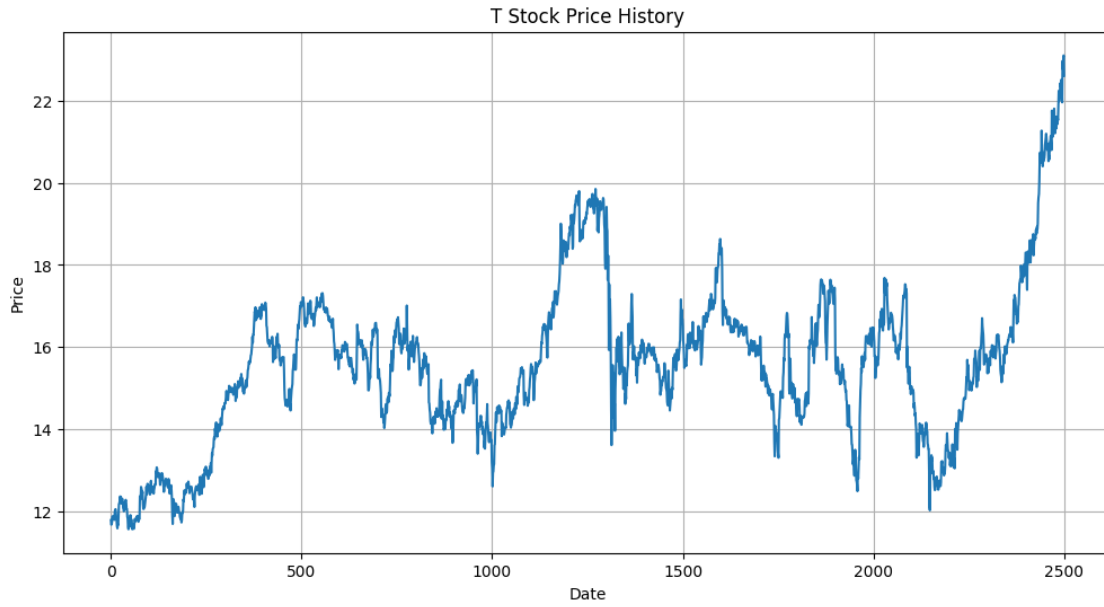
dtype: int64

Key Metrics:

Total Trading Days: 2501

Average Daily Volume: 42,610,299

Average Daily Return: 0.0363%
Return Volatility: 1.4320%
Maximum Daily Gain: 10.0223%
Maximum Daily Loss: -10.4061%



Analysis for UNH:

Dataset Info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2501 entries, 0 to 2500

Data columns (total 10 columns):

#	Column	Non-Null Count	Dtype
0	Date	2501 non-null	datetime64[ns]
1	Open	2501 non-null	float64
2	High	2501 non-null	float64
3	Low	2501 non-null	float64
4	Close	2501 non-null	float64
5	Volume	2501 non-null	int64
6	Dividends	2501 non-null	float64
7	Stock Splits	2501 non-null	float64
8	Ticker	2501 non-null	object
9	Daily_Return	2500 non-null	float64

dtypes: datetime64[ns](1), float64(7), int64(1), object(1)

memory usage: 195.5+ KB

None

Summary Statistics:

	Date	Open	High	Low
count	2501	2501.000000	2501.000000	2501.000000
mean	2019-12-20 21:48:08.924430336	296.731172	299.646057	293.766347
min	2015-01-02 00:00:00	84.039447	85.041543	81.289273
25%	2017-06-27 00:00:00	164.329372	164.851555	163.037302
50%	2019-12-19 00:00:00	248.542148	252.188431	245.678459
75%	2022-06-14 00:00:00	462.404920	467.986044	458.686578
max	2024-12-09 00:00:00	610.254097	621.236179	604.679207
std	NaN	152.554260	154.004666	151.096066

	Close	Volume	Dividends	Stock Splits	Daily_Return
count	2501.000000	2.501000e+03	2501.000000	2501.0	2500.000000
mean	296.771394	3.570029e+06	0.018798	0.0	0.088239
min	84.005486	7.140000e+05	0.000000	0.0	-17.276868
25%	163.966492	2.529800e+06	0.000000	0.0	-0.686153
50%	250.104202	3.102900e+06	0.000000	0.0	0.085932
75%	462.578156	4.033500e+06	0.000000	0.0	0.834174
max	615.838684	2.736140e+07	2.100000	0.0	12.798922
std	152.534865	1.870296e+06	0.161112	0.0	1.641065

Missing Values:

Daily_Return 1

dtype: int64

Key Metrics:

Total Trading Days: 2501

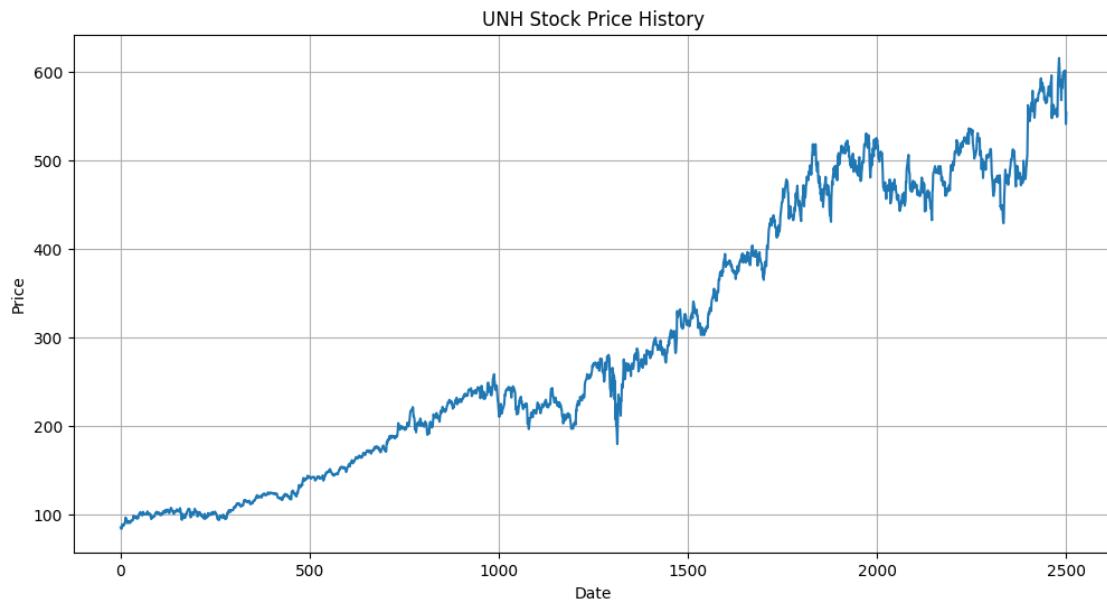
Average Daily Volume: 3,570,029

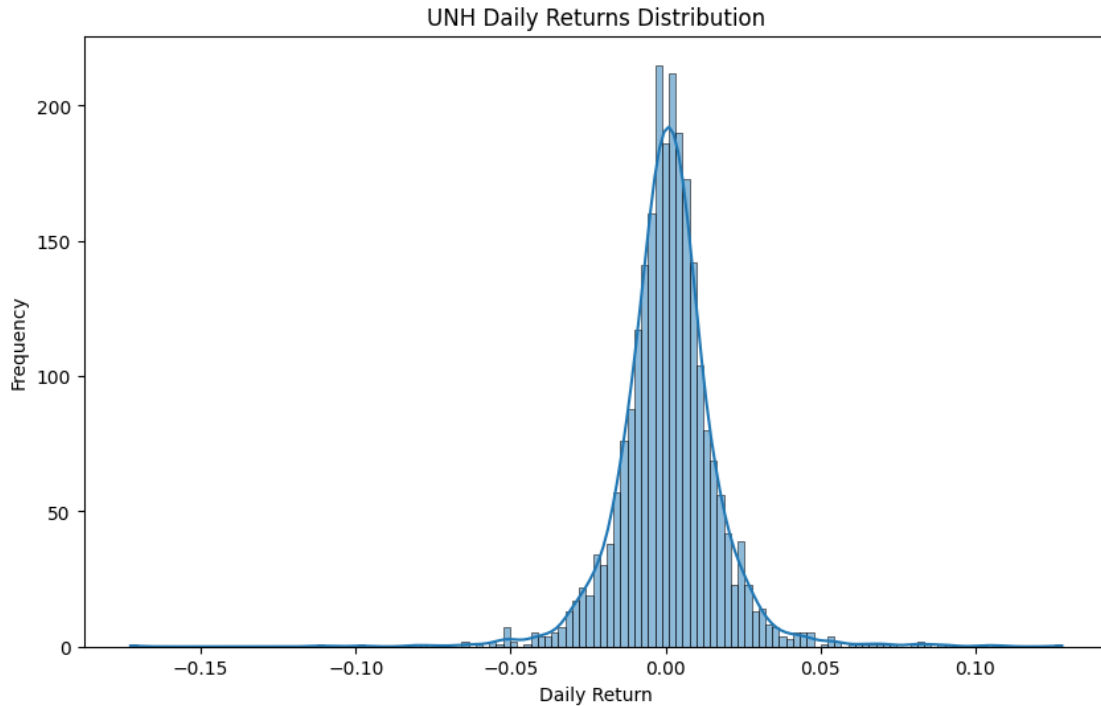
Average Daily Return: 0.0882%

Return Volatility: 1.6411%

Maximum Daily Gain: 12.7989%

Maximum Daily Loss: -17.2769%





```
[21]: warnings.filterwarnings("ignore", category=UserWarning)
       warnings.filterwarnings("ignore", category=FutureWarning)
```

2 Efficient Portfolios

First, we prepare our data and some functions used to find efficient portfolios

```
[22]: # Set the in-sample dataset
START_DATE = '2015-01-01'
END_DATE = '2022-12-12'
YEAR = 252 #number of trading days

stock_returns.index = pd.to_datetime(stock_returns.index)

#create the dataframe used for all portfolio construction:
stock_returns_subset = stock_returns[(stock_returns.index >= START_DATE) &
                                     (stock_returns.index <= END_DATE)]

insample = len(stock_returns_subset)
daily_expected_returns = stock_returns_subset.mean()
assets = len(daily_expected_returns)
print(f" For in-sample data, we have returns for {insample} days.")
```

For in-sample data, we have returns for 2000 days.

```
[23]: # calculates CVaR of individual assets
def asset_cvar(asset_returns: pd.Series, alpha: float):
    var_threshold = np.quantile(asset_returns, 1 - alpha)
    cvar = -asset_returns[asset_returns <= var_threshold].mean()
    return cvar
```

```
[24]: #summary for all data, not only insample
cvar_values = []
for col in stock_returns.columns:
    cvar = asset_cvar(stock_returns[col], 0.9)
    cvar_values.append(cvar)

summary = pd.DataFrame({
    'Mean': stock_returns.mean(),
    'Min': stock_returns.min(),
    'Max': stock_returns.max(),
    'Variance': stock_returns.var(),
    'CVaR_90': cvar_values,
    'Ratio': stock_returns.var()/cvar_values
})

print(summary)
```

	Mean	Min	Max	Variance	CVaR_90	Ratio
AAPL	0.108813	-12.864693	11.980810	3.227022	3.191256	1.011207
AMZN	0.128696	-14.049438	14.131126	4.271864	3.559504	1.200129
BAC	0.065259	-15.397343	17.796180	3.858294	3.384235	1.140079
COST	0.095437	-12.451340	9.959463	1.852109	2.360158	0.784739
IBM	0.044387	-12.850751	11.301055	2.254734	2.674233	0.843133
JNJ	0.031869	-10.037886	7.997706	1.294897	2.000490	0.647290
JPM	0.080131	-14.964867	18.012479	2.981475	2.948285	1.011257
KO	0.034881	-9.672484	6.479564	1.258562	2.008008	0.626771
T	0.036349	-10.406097	10.022340	2.050486	2.519906	0.813715
UNH	0.088239	-17.276868	12.798922	2.693094	2.757615	0.976603

```
[25]: #calculates CVaR and return given an array of portfolio weights
def calculate_portfolio_return_and_cvar(returns_df, weights_array, alpha):
    # Daily returns
    portfolio_returns = returns_df @ weights_array

    # Expected return
    mean_return = portfolio_returns.mean()

    # VaR
    var_threshold = np.quantile(portfolio_returns, 1 - alpha)

    # CVaR
    cvar = -portfolio_returns[portfolio_returns <= var_threshold].mean()
```

```
return mean_return, cvar
```

```
[26]: #generates grid of weights that combine only two assets so that we can find  
↳lowest feasible CVaR values
```

```
def generate_feasible_weights(tickers, grid_steps=100):  
    n_assets = len(tickers)  
    weights_list = []  
  
    for i, j in itertools.combinations(range(n_assets), 2):  
        grid = []  
        for k in np.linspace(0, 1, grid_steps):  
            w = np.zeros(n_assets)  
            w[i] = k  
            w[j] = 1 - k  
            grid.append(w)  
        weights_list.append(np.array(grid))  
  
    return np.vstack(weights_list)
```

```
[27]: #generate the weights  
weights_all = generate_feasible_weights(tickers=stock_returns_subset.columns)
```

```
results = []  
for weights in weights_all:  
    try:  
        ret, cvar = calculate_portfolio_return_and_cvar(stock_returns_subset, ↳  
↳weights, alpha=0.9)  
        _, cvar2 = calculate_portfolio_return_and_cvar(stock_returns_subset, ↳  
↳weights, alpha=0.995)  
        results.append({'Return': ret, 'CVaR': cvar, 'CVaR2': cvar2})  
    except Exception as e:  
        print(f"Error with weights {weights}: {e}")
```

```
[28]: # prepare data for plotting  
df_results = pd.DataFrame(results)  
df_results['Return_round'] = df_results['Return'].round(3)  
worst_cvar_frontier = df_results.loc[df_results.groupby('Return_round')['CVaR'].  
↳idxmax()]
```

```
worst_cvar_frontier = worst_cvar_frontier.sort_values('Return')
```

```
[29]: # finds min CVaR_alpha portfolio that also minimizes the other values if ↳  
↳possible, using the methods described in the thesis  
# requires return data and level alpha  
# target return or target secondary CVaR value and level can be also set
```

```

def min_cvar_portfolio(returns: pd.DataFrame, alpha: float, target_return:
↳float | None = None, alpha2: float | None = None, target_cvar2: float | None
↳= None):

    R = returns.values
    T, n = R.shape

    # Variables
    w = cp.Variable(n)
    z = cp.Variable()
    u = cp.Variable(T)

    # Objective:  $z + (1 / (T*(1-\alpha))) * \text{sum}(u)$ 
    obj = cp.Minimize(z + cp.sum(u) / (T * (1 - alpha)))

    # Constraints:
    constraints = [
        cp.sum(w) == 1,
        w >= 0,
        u >= 0,
        u >= -(R @ w) - z
    ]

    # Optional return constraint
    if target_return is not None:
        mu = returns.mean().values
        constraints.append(w @ mu >= target_return)

    # optional CVaR_alpha2 constraint
    if (alpha2 is None) ^ (target_cvar2 is None):
        raise ValueError("Specify both alpha2 and target_cvar2 together")
    if alpha2 is not None:
        z2 = cp.Variable()
        u2 = cp.Variable(T)
        constraints += [
            u2 >= 0,
            u2 >= -(R @ w) - z2
        ]
        cvar2 = z2 + cp.sum(u2) / (T * (1 - alpha2))
        constraints.append(cvar2 <= target_cvar2)

    prob = cp.Problem(obj, constraints)
    prob.solve(solver=cp.ECOS)

    # Results
    weights = w.value

```

```

losses = -R @ weights
var_threshold = np.quantile(losses, alpha)
optimal_cvar = var_threshold + np.mean(np.maximum(losses - var_threshold,
↪0)) / (1 - alpha)
cvar2=0
if alpha2 is not None:
    var_threshold2 = np.quantile(losses, alpha2)
    cvar2 = var_threshold2 + np.mean(np.maximum(losses - var_threshold2,
↪0)) / (1 - alpha2)
exp_return = np.dot(returns.mean(), weights)

return weights, optimal_cvar, exp_return, cvar2

```

```

[30]: # summary for in-sample
cvar_values = []
for col in stock_returns_subset.columns:
    cvar = asset_cvar(stock_returns_subset[col], 0.9)
    cvar_values.append(cvar)

summary = pd.DataFrame({
    'Mean': stock_returns_subset.mean(),
    'Min': stock_returns_subset.min(),
    'Max': stock_returns_subset.max(),
    'CVaR_90': cvar_values,
})

print(summary)

```

	Mean	Min	Max	CVaR_90
AAPL	0.106311	-12.864693	11.980810	3.359423
AMZN	0.110370	-14.049438	14.131126	3.652584
BAC	0.058825	-15.397343	17.796180	3.563598
COST	0.080374	-12.451340	9.959463	2.450723
IBM	0.027622	-12.850751	11.301055	2.808749
JNJ	0.044210	-10.037886	7.997706	2.056107
JPM	0.065250	-14.964867	18.012479	3.068438
KO	0.040775	-9.672484	6.479564	2.135118
T	0.026745	-9.241038	10.022340	2.514076
UNH	0.104668	-17.276868	12.798922	2.775763

2.0.1 Benchmark model (M1)

```

[31]: # weights and performance of efficient portfolios for the benchmark model
target_returns = np.linspace(stock_returns_subset.mean().min(),
↪stock_returns_subset.mean().max(), 100)

frontier_data = []

```

```

weights_data = []

for tr in target_returns:
    try:
        w, cvar, ret, _ = min_cvar_portfolio(
            returns=stock_returns_subset,
            alpha=0.9,
            target_return=tr
        )
        frontier_data.append({'Target_Return': tr, 'Return': ret, 'CVaR': cvar})
        weights_data.append(w)
    except Exception as e:
        print('infeasible return')
        continue

frontier_df1 = pd.DataFrame(frontier_data)
weights_df1 = pd.DataFrame(weights_data, index=target_returns,
    ↪ columns=stock_returns_subset.columns)

fig, ax = plt.subplots(figsize=(8, 4))

weights_df1.plot(ax=ax, linewidth=1.5)

ax.set_ylabel("Portfolio Weight")
ax.set_xlabel("Daily Expected Return (%)")
ax.tick_params(axis='x', labelsize=8, rotation=0)
ax.tick_params(axis='y', labelsize=8)

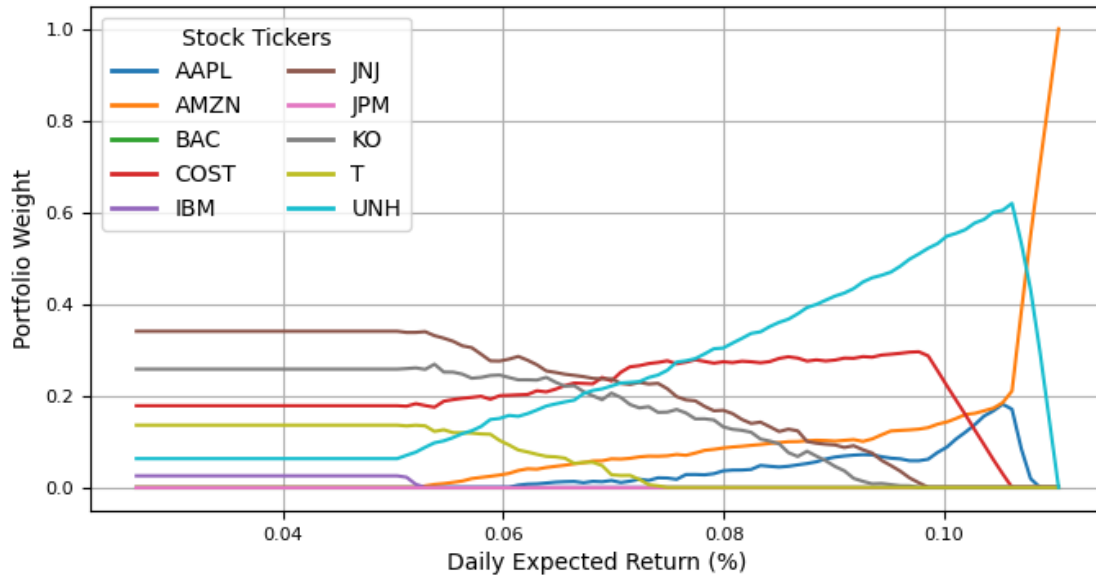
for label in ax.get_xticklabels():
    label.set_horizontalalignment('center')

ax.grid(True)

legend = ax.legend(title="Stock Tickers", loc="upper left", ncol=2)
for line in legend.get_lines():
    line.set_linewidth(2.0)

plt.savefig("weights_vs_return.pdf", dpi=300, bbox_inches='tight',
    ↪ transparent=True)
plt.show()

```



```
[32]: # plot of efficient set for the benchamrk model
# part of the code can be uncommented for the feasible set to work, however,
# first we must adjust the min_cvar_portfolio function to be able to find
# portfolios that minimize CVaR with lower returns (left bottom part of the
# plot)
fig, ax = plt.subplots(figsize=(8, 4))

# Plot feasible set (random portfolios)
plt.plot(worst_cvar_frontier['CVaR'], worst_cvar_frontier['Return'],
         color='gray', linewidth=2, label='_nolegend_')

min_cvar_idx = frontier_df1['CVaR'].idxmin()
min_cvar_point = frontier_df1.loc[min_cvar_idx]

low_return = frontier_df1[frontier_df1['Return'] < min_cvar_point['Return']]
high_return = frontier_df1[frontier_df1['Return'] >= min_cvar_point['Return']]

high_return.sort_values('Return').plot(
    x='CVaR', y='Return', ax=ax, color='blue', linewidth=2, label='Efficient
    Portfolios'
)

low_return.sort_values('Return').plot(
    x='CVaR', y='Return', ax=ax, color='gray', linewidth=2, label='Feasible set
    boundary'
)
```

```

# Plot individual assets
ax.scatter(
    summary['CVaR_90'],
    summary['Mean'],
    color='red',
    marker='x',
    s=80,
    label='Individual Assets'
)

# # Create a shared return grid (intersection or a fine grid within overlap)
# common_returns = np.linspace(
#     max(frontier_df1_sorted['Return'].min(), worst_sorted['Return'].min()),
#     min(frontier_df1_sorted['Return'].max(), worst_sorted['Return'].max()),
#     300
# )

# # Interpolate CVaR values along the shared return grid
# best_interp_cvar = np.interp(common_returns, frontier_df1_sorted['Return'],
#     ↪frontier_df1_sorted['CVaR'])
# worst_interp_cvar = np.interp(common_returns, worst_sorted['Return'],
#     ↪worst_sorted['CVaR'])

# # Fill the area between curves
# ax.fill_betweenx(
#     common_returns,
#     best_interp_cvar,
#     worst_interp_cvar,
#     color='lightgray',
#     alpha=0.4,
#     label='Feasible Region'
# )

# Compute median return to decide placement
median_return = summary['Mean'].median()

for ticker in summary.index:
    x = summary['CVaR_90'][ticker]
    y = summary['Mean'][ticker]

    if y > median_return:
        va = 'top'
        offset = (0, -5)
    else:
        va = 'bottom'
        offset = (0, 5)

```

```

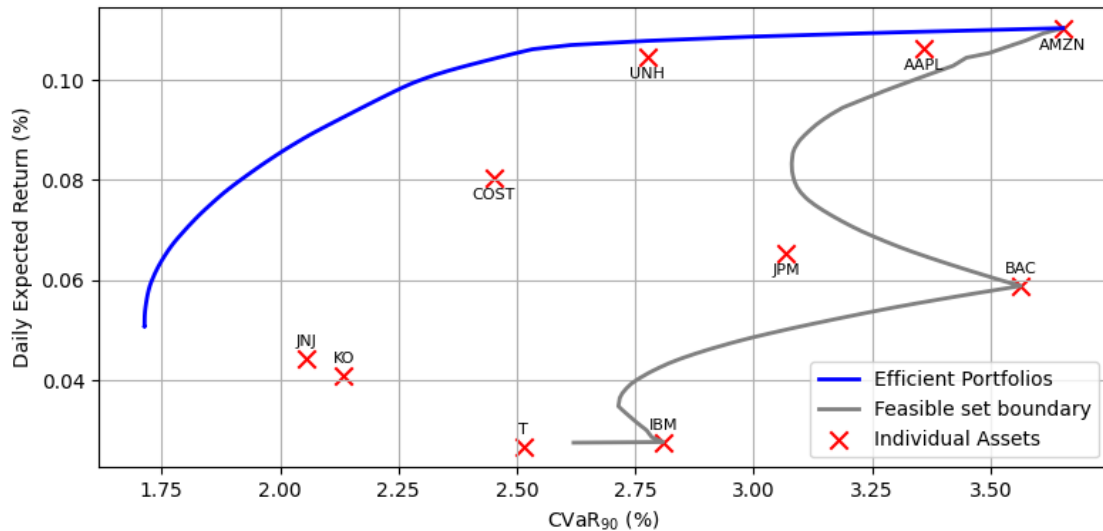
ax.annotate(ticker, (x, y),
            textcoords="offset points", xytext=offset,
            ha='center', va=va, fontsize=8)

frontier_df1_sorted = frontier_df1.sort_values('Return')
worst_sorted = worst_cvar_frontier.sort_values('Return')

# Labels and formatting
ax.set_xlabel(" $\text{CVaR}_{90}$  (%)")
ax.set_ylabel("Daily Expected Return (%)")
ax.legend()
ax.grid(True)
plt.tight_layout()

plt.savefig("benchmark.pdf", dpi=300, bbox_inches='tight', transparent=True)
plt.show()

```



```

[33]: # Finds mean-variance efficient set for comparison
def mean_variance_portfolio(returns, target_return):
    mean_returns = returns.mean().values
    cov_matrix = returns.cov().values
    num_assets = len(mean_returns)

    # Objective: Minimize portfolio variance
    def portfolio_variance(w):
        return w.T @ cov_matrix @ w

    # Constraints

```

```

constraints = [
    {'type': 'eq', 'fun': lambda w: np.sum(w) - 1}, # Weights sum to 1
    {'type': 'eq', 'fun': lambda w: w @ mean_returns - target_return} #
↳Target return
]

bounds = [(0, 1)] * num_assets
w0 = np.ones(num_assets) / num_assets

# Optimization
result = minimize(portfolio_variance, w0, bounds=bounds,
↳constraints=constraints)

if not result.success:
    raise ValueError("Optimization failed: " + result.message)

w_opt = result.x
var_opt = portfolio_variance(w_opt)
ret_opt = w_opt @ mean_returns

return w_opt, var_opt, ret_opt

```

```

[34]: def l1_distance(w1, w2):
    return np.sum(np.abs(np.array(w1) - np.array(w2)))

```

```

[35]: def max_drawdown(cum_returns):
    peaks = np.maximum.accumulate(cum_returns)
    drawdowns = (peaks - cum_returns) / peaks
    return drawdowns.max()

```

```

[36]: # creation of latex table of chosen benchmark portfolios and save performance
↳data in 'M1_results_df'
target_returns = [stock_returns_subset.mean().min(), 0.06, 0.085, 0.105,
↳stock_returns_subset.mean().max()]

frontier_data = []
weights_data = []
l1_distances = []

for tr in target_returns:
    try:
        w, cvar, ret, _ = min_cvar_portfolio(
            returns=stock_returns_subset,
            alpha=0.9,
            target_return=tr
        )
        w_mark, var_mark, ret_mark = mean_variance_portfolio(

```

```

        returns=stock_returns_subset,
        target_return=ret
    )
    distance = l1_distance(w, w_mark)
    port_rets = stock_returns_subset @ w
    cum = (1 + port_rets/100).cumprod()
    single_day_max_loss = -port_rets.values.min()
    max_dd = max_drawdown(cum.values)*100

    # Store all metrics
    frontier_data.append({
        'Target_Return': tr,
        'Return': ret,
        'CVaR': cvar,
        'MaxDailyLoss': single_day_max_loss,
        'MaxDrawdown': max_dd,
        'Weights': w
    })
    weights_data.append(w)
    l1_distances.append(distance)
except Exception as e:
    print(f'infeasible return: {tr}')
    continue

benchmark_weights = weights_data.copy()

table_df = pd.DataFrame(frontier_data)
table_weights_df = pd.DataFrame(weights_data, index=[f"{tr:.2f}" for tr in
↳table_df['Target_Return']],
                                columns=stock_returns_subset.columns)

M1_result_df = pd.DataFrame(frontier_data)

threshold = 1e-3
selected_labels = [f"Return={tr:.2f}" for tr in table_df['Target_Return']]
weights_subset = table_weights_df
weights_subset.index = selected_labels
weights_transposed = weights_subset.T

def format_weight(val):
    return f"{val:.3f}" if abs(val) >= threshold else ""

weights_transposed = weights_transposed.applymap(format_weight)
weights_transposed = weights_transposed.reset_index()
weights_transposed.columns = ['Ticker'] + selected_labels

# CVaR, Return, and Distance rows

```

```

cvar_row = ["CVaR"] + [f"{v:.3f}" for v in table_df['CVaR']]
return_row = ["Return"] + [f"{v:.3f}" for v in table_df['Return']]
distance_row = ["L1 Distance"] + [f"{v:.3f}" for v in l1_distances]

column_format = 'l' + 'r' * len(selected_labels)

latex_table = (
    "\\begin{table}[b]\n"
    "\\centering\n"
    f"\\begin{tabular}{{{column_format}}}\n"
    "\\toprule\n"
    "\\textbf{Ticker} & " + " & ".join(f"\\textbf{{{col}}}" for col in_
↪selected_labels) + " \\\\n"
    "\\midrule\n"
    "\\textbf{CVaR} & " + " & ".join(cvar_row[1:]) + " \\\\n"
    "\\textbf{Return} & " + " & ".join(return_row[1:]) + " \\\\n"
    "\\midrule\n"
)

for _, row in weights_transposed.iterrows():
    values = " & ".join(row[col] if row[col] != "" else " " for col in_
↪selected_labels)
    latex_table += f"{row['Ticker']} & {values} \\\\n"

# Add distance row
latex_table += "\\midrule\n"
latex_table += "\\textbf{L1 Distance} & " + " & ".join(distance_row[1:]) + "
↪\\\\n"

latex_table += (
    "\\bottomrule\n"
    "\\end{tabular}\n"
    "\\caption{Portfolio weights for selected target returns, with CVaR,
↪return, and distance from Markowitz portfolios (weights < 0.001 omitted)}\n"
    "\\label{tab:portfolio_weights_custom_returns}\n"
    "\\end{table}"
)

print(latex_table)

```

```

\begin{table}[b]
\centering
\begin{tabular}{lrrrrr}
\toprule
\textbf{Ticker} & \textbf{Return=0.03} & \textbf{Return=0.06} &
\textbf{Return=0.09} & \textbf{Return=0.10} & \textbf{Return=0.11} \\
\midrule

```

```

\textbf{CVaR} & 1.714 & 1.730 & 1.993 & 2.482 & 3.653 \\
\textbf{Return} & 0.051 & 0.060 & 0.085 & 0.105 & 0.110 \\
\midrule
AAPL & & & 0.044 & 0.179 & \\
AMZN & & & 0.029 & 0.098 & 0.180 & 1.000 \\
BAC & & & & & \\
COST & 0.178 & 0.200 & 0.281 & 0.041 & \\
IBM & 0.025 & & & & \\
JNJ & 0.341 & 0.279 & 0.122 & & \\
JPM & & & & & \\
KO & 0.258 & 0.245 & 0.095 & & \\
T & 0.136 & 0.096 & & & \\
UNH & 0.063 & 0.151 & 0.359 & 0.600 & \\
\midrule
\textbf{L1 Distance} & 0.172 & 0.149 & 0.171 & 0.219 & 0.000 \\
\bottomrule
\end{tabular}
\caption{Portfolio weights for selected target returns, with CVaR, return, and distance from Markowitz portfolios (weights < 0.001 omitted)}
\label{tab:portfolio_weights_custom_returns}
\end{table}

```

2.0.2 Extreme-tail model (M2)

```

[37]: #calculates efficient set for extreme-tail model (M2) at specified value of
      ↪ CVaR_99.5 constraint
n=50
cvar2_constaint = 5.7 # CVaR_99.5 is between 5.158 and 8.215

target_returns = np.linspace(0.05, stock_returns_subset.mean().max(), n)

new_frontier_data = []
new_weights_data = []
counter=0

for tr in target_returns:
    try:
        w, cvar, ret, cvar2 = min_cvar_portfolio(
            returns=stock_returns_subset,
            alpha=0.9,
            target_return=tr,
            alpha2=0.995,
            target_cvar2=cvar2_constaint
        )
        new_frontier_data.append({'Target_Return': tr, 'Return': ret, 'CVaR':
            ↪ cvar})
        new_weights_data.append(w)

```

```

except Exception:
    counter+=1
    continue

frontier_new = pd.DataFrame(new_frontier_data)
weights_new = pd.DataFrame(new_weights_data, columns=stock_returns_subset.
    ↪columns)
weights_new.index = frontier_new['Target_Return'].values

```

```

[38]: #generates again weights of benchmark portfolios to compare against new model
target_returns = np.linspace(0.05, stock_returns_subset.mean().max(), 100)

old_frontier_data = []
old_weights_data = []

for tr in target_returns:
    try:
        w, cvar, ret, _ = min_cvar_portfolio(
            returns=stock_returns_subset,
            alpha=0.9,
            target_return=tr
        )
        old_frontier_data.append({'Target_Return': tr, 'Return': ret, 'CVaR':_
    ↪cvar})
        old_weights_data.append(w)
    except Exception as e:
        print('infeasible return')
        continue

frontier_old = pd.DataFrame(old_frontier_data)
weights_old = pd.DataFrame(old_weights_data, index=target_returns,
    ↪columns=stock_returns_subset.columns)

```

```

[39]: fig, ax = plt.subplots(figsize=(8, 4))

weights_old.columns = ['_' + col for col in weights_old.columns]
# Plot baseline weight lines with dashed style and low alpha
weights_old.plot(ax=ax,
                 linestyle='--',
                 alpha=0.9,
                 linewidth=1)

# Overlay new weight lines from constrained frontier
weights_new.plot(ax=ax,
                 linestyle='-',
                 alpha=1.0,
                 linewidth=1.5)

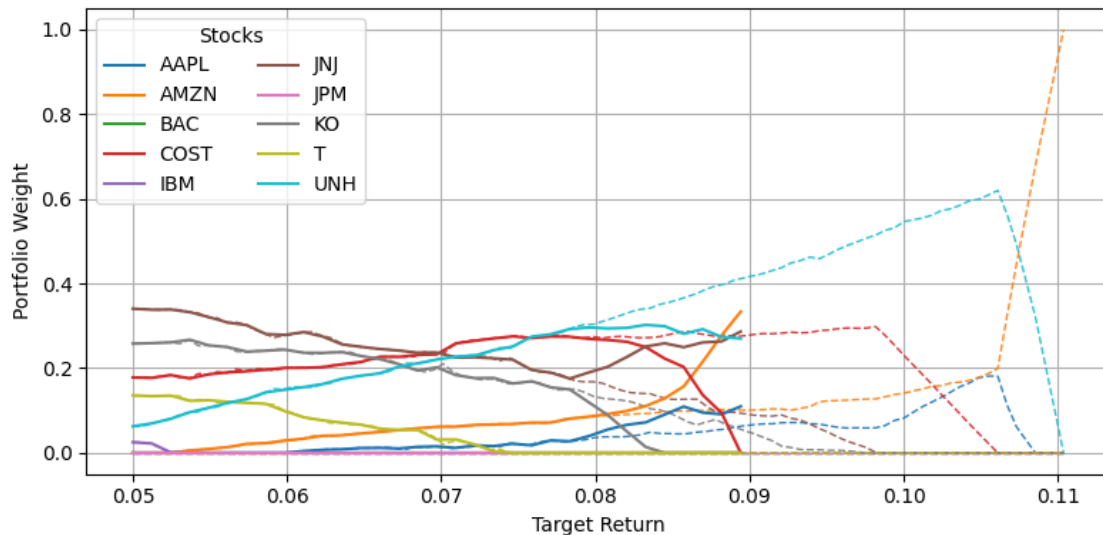
```

```

ax.set_xlabel('Target Return')
ax.set_ylabel('Portfolio Weight')
ax.grid(True)
ax.legend(title="Stocks", ncol=2)
plt.tight_layout()

plt.savefig("extreme-tail_weights.pdf", dpi=300, bbox_inches='tight',
           transparent=True)
plt.show()

```



```

[40]: #calculates efficient sets for multiple CVaR_99.5 constraints, requires
      ↪relatively high n
cvar2_constraints = [5.16, 5.2, 5.3, 5.471, 5.6, 6, 6.5, None] # None = no
      ↪constraint
frontiers = {}

n = 250
target_returns = np.linspace(stock_returns_subset.mean().min(),
      ↪stock_returns_subset.mean().max(), n)

for constraint in cvar2_constraints:
    records = []
    for tr in target_returns:
        try:
            w, cvar, ret, cvar2 = min_cvar_portfolio(
                returns=stock_returns_subset,

```

```

        alpha=0.9,
        target_return=tr,
        alpha2=0.995,
        target_cvar2=constraint if constraint is not None else 1e6 #
↳effectively unconstrained
    )
    records.append({'Target_Return': tr, 'Return': ret, 'CVaR': cvar,
↳'CVaR2': cvar2, 'Weights': w})
    except Exception:
        continue

    # Create DataFrame for this constraint level
    frontier_df = pd.DataFrame([
        {'Target_Return': rec['Target_Return'], 'Return': rec['Return'], 'CVaR':
↳ rec['CVaR'], 'CVaR2': rec['CVaR2']}
        for rec in records
    ])
    frontiers[constraint] = frontier_df

```

```

[41]: #calculates portfolios that minimize CVaR_99.5 to show the edge of the plot
target_returns = np.linspace(stock_returns_subset.mean().min(),
                             stock_returns_subset.mean().max(), 100)

frontier_cv2 = []
weights_cv2 = []

for tr in target_returns:
    try:
        w, cvar, ret, _ = min_cvar_portfolio(
            returns=stock_returns_subset,
            alpha=0.995,
            target_return=tr
        )
        _, cvar90 =
↳calculate_portfolio_return_and_cvar(returns_df=stock_returns_subset,
↳weights_array=w, alpha=0.9)
        frontier_cv2.append({'Target_Return': tr, 'Return': ret, 'CVaR': cvar,
↳'CVaR90': cvar90})
        weights_cv2.append(w)
    except Exception as e:
        print('infeasible return')
        continue

frontier_dfcv2 = pd.DataFrame(frontier_cv2)
weights_dfcv2 = pd.DataFrame(weights_cv2, index=target_returns,
↳columns=stock_returns_subset.columns)

```

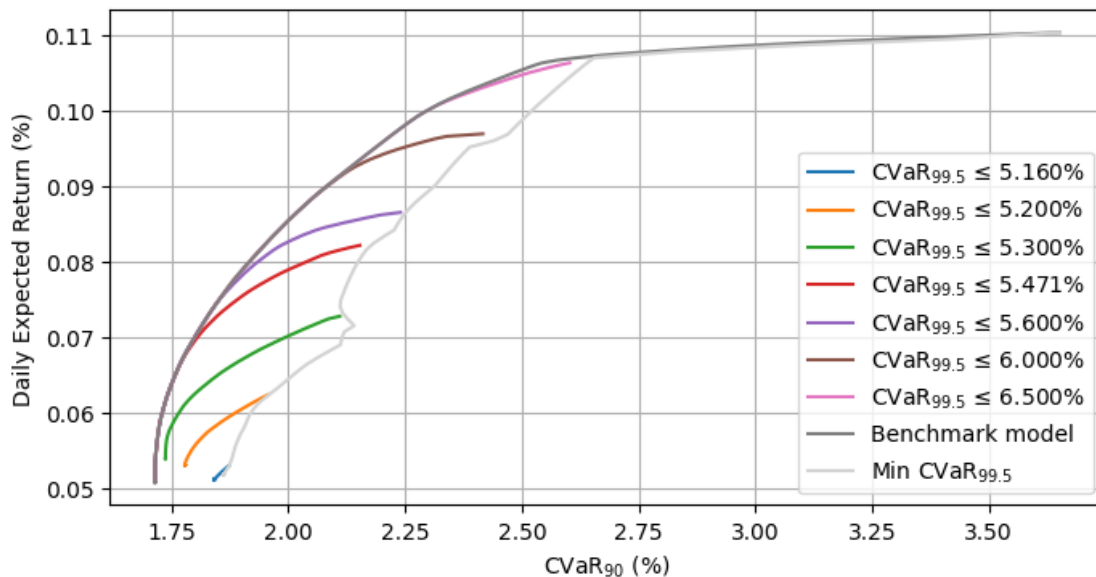
```
[43]: plt.figure(figsize=(8, 4))

for constraint, df in frontiers.items():
    label = f"CVaR$_{{99.5}}$ {constraint:.3f}%" if constraint is not None
    ↪else "Benchmark model"
    marker = 'o' if len(df) == 1 else None # Show single point
    plt.plot(df['CVaR'], df['Return'], label=label)

plt.plot(frontier_dfcv2['CVaR90'], frontier_dfcv2['Return'], label="Min_
↪CVaR$_{{99.5}}$", marker=marker,color='lightgray')

plt.xlabel("$\mathregular{CVaR_{90}}$ (%)")
plt.ylabel("Daily Expected Return (%)")
plt.legend()
plt.grid(True)

plt.savefig("extreme-tail.pdf", dpi=300, bbox_inches='tight', transparent=True)
plt.show()
```



```
[44]: # Create a latex table of chosen set of portfolios, store for comparison as_
↪'M2_results_df'
target_returns = [0.06, 0.085, 0.105]
cvar2_constraints = [5.25, 5.7, 6.5]

frontier_data = []
weights_data = []
l1_distances = []
```

```

for i, tr in enumerate(target_returns):
    try:
        w, cvar, ret, _ = min_cvar_portfolio(
            returns=stock_returns_subset,
            alpha=0.9,
            target_return=tr,
            alpha2=0.995,
            target_cvar2=cvar2_constraints[i]
        )
        distance = l1_distance(w, benchmark_weights[i+1])
        port_rets = stock_returns_subset @ w
        cum = (1 + port_rets/100).cumprod()
        single_day_max_loss = -port_rets.values.min()
        max_dd = max_drawdown(cum.values)*100
        frontier_data.append({'Target_Return': tr, 'Return': ret, 'CVaR': cvar,
↪ 'MaxDailyLoss': single_day_max_loss, 'MaxDrawdown': max_dd, 'Weights': w})
        weights_data.append(w)
        l1_distances.append(distance)
    except Exception as e:
        print(f'infeasible return: {tr}')
        continue

table_df = pd.DataFrame(frontier_data)
table_weights_df = pd.DataFrame(weights_data, index=[f"{tr:.2f}" for tr in
↪ table_df['Target_Return']],
                                columns=stock_returns_subset.columns)

M2_result_df = pd.DataFrame(frontier_data)

threshold = 1e-3
selected_labels = [f"Return={tr:.2f}" for tr in table_df['Target_Return']]
weights_subset = table_weights_df
weights_subset.index = selected_labels
weights_transposed = weights_subset.T

def format_weight(val):
    return f"{val:.3f}" if abs(val) >= threshold else ""

weights_transposed = weights_transposed.applymap(format_weight)
weights_transposed = weights_transposed.reset_index()
weights_transposed.columns = ['Ticker'] + selected_labels

# CVaR, Return, and Distance rows
cvar_row = ["CVaR"] + [f"{v:.3f}" for v in table_df['CVaR']]

```

```

return_row = ["Return"] + [f"{v:.3f}" for v in table_df['Return']]
distance_row = ["L1 Distance"] + [f"{v:.3f}" for v in l1_distances]

column_format = 'l' + 'r' * len(selected_labels)

latex_table = (
    "\\begin{table}[b]\n"
    "\\centering\n"
    f"\\begin{tabular}{{{column_format}}}\n"
    "\\toprule\n"
    "\\textbf{Ticker} & " + " & ".join(f"\\textbf{{{col}}}" for col in
↪selected_labels) + " \\\\n"
    "\\midrule\n"
    "\\textbf{CVaR} & " + " & ".join(cvar_row[1:]) + " \\\\n"
    "\\textbf{Return} & " + " & ".join(return_row[1:]) + " \\\\n"
    "\\midrule\n"
)

for _, row in weights_transposed.iterrows():
    values = " & ".join(row[col] if row[col] != "" else " " for col in
↪selected_labels)
    latex_table += f"{row['Ticker']} & {values} \\\\n"

latex_table += "\\midrule\n"
latex_table += "\\textbf{L1 Distance} & " + " & ".join(distance_row[1:]) + "
↪\\\n"

latex_table += (
    "\\bottomrule\n"
    "\\end{tabular}\n"
    "\\caption{Portfolio weights for selected target returns, with CVaR,
↪return, and distance from Markowitz portfolios (weights < 0.001 omitted)}\n"
    "\\label{tab:portfolio_weights_custom_returns}\n"
    "\\end{table}"
)

print(latex_table)

```

```

\begin{table}[b]
\centering
\begin{tabular}{lrrr}
\toprule
\textbf{Ticker} & \textbf{Return=0.06} & \textbf{Return=0.09} &
\textbf{Return=0.10} \\
\midrule
\textbf{CVaR} & 1.811 & 2.022 & 2.514 \\

```

```

\textbf{Return} & 0.060 & 0.085 & 0.105 \\
\midrule
AAPL & & 0.101 & 0.190 \\
AMZN & 0.146 & 0.138 & 0.276 \\
BAC & & & \\
COST & 0.168 & 0.209 & 0.064 \\
IBM & & & \\
JNJ & 0.448 & 0.257 & \\
JPM & & & \\
KO & 0.036 & & \\
T & 0.154 & & \\
UNH & 0.048 & 0.294 & 0.470 \\
\midrule
\textbf{L1 Distance} & 0.689 & 0.464 & 0.259 \\
\bottomrule
\end{tabular}
\caption{Portfolio weights for selected target returns, with CVaR, return, and
distance from Markowitz portfolios (weights < 0.001 omitted)}
\label{tab:portfolio_weights_custom_returns}
\end{table}

```

2.0.3 General mean-CVaR model

```

[45]: #comparison of efficient sets for different CVaR levels
alphas = [0.80, 0.90, 0.95, 0.99, 0.995, 0.999]
target_returns = np.linspace(stock_returns_subset.mean().min(),
↪stock_returns_subset.mean().max(), 100)

all_frontiers = {}
all_weights = {}

for alpha in alphas:
    frontier_list = []
    weights_list = []
    for tr in target_returns:
        try:
            w, cvar, ret, _ = min_cvar_portfolio(
                returns=stock_returns_subset,
                alpha=alpha,
                target_return=tr
            )
            _, cvar90 = ↪
↪calculate_portfolio_return_and_cvar(returns_df=stock_returns_subset,
↪weights_array=w, alpha=0.9)
            frontier_list.append({'Target_Return': tr, 'Return': ret, 'CVaR': ↪
↪cvar, 'CVaR90': cvar90})
            weights_list.append(w)

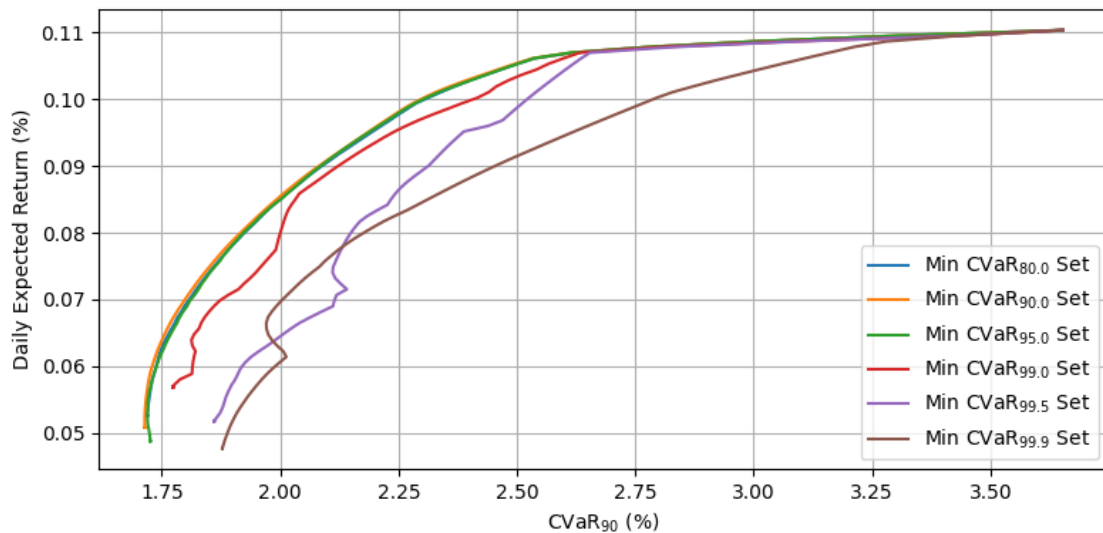
```

```
except Exception:
    continue
```

```
df_frontier = pd.DataFrame(frontier_list)
df_weights = pd.DataFrame(weights_list, columns=stock_returns_subset.
↳columns)
df_weights.index = df_frontier['Target_Return'].values

all_frontiers[alpha] = df_frontier
all_weights[alpha] = df_weights
```

```
[46]: plt.figure(figsize=(8, 4))
for alpha, df in all_frontiers.items():
    label = f"Min CVaR${int(alpha*1000)/10:.1f}$ Set"
    plt.plot(df['CVaR90'], df['Return'], label=label)
plt.xlabel("$\mathregular{CVaR_{90}}$ (%)")
plt.ylabel("Daily Expected Return (%)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("multiple_frontiers.pdf", dpi=300, bbox_inches='tight',
↳transparent=True)
plt.show()
```



2.0.4 Model M3

This model considers CVaR₉₀ and CVaR₉₅.

```

[47]: # Find the "boundary" of efficient set for M3
base = np.linspace(stock_returns_subset.mean().min(), stock_returns_subset.
↳mean().max(), 100)
top5 = np.sort(base)[-5:] # pick top 5 highest
extra_ranges = [np.linspace(top5[i], top5[i+1], 3, endpoint=False)[1:]
↳for i in range(4)]
extra = np.concatenate(extra_ranges)
target_returns = np.sort(np.unique(np.concatenate([base, extra])))
a1=0.9
a2=0.95

#mean-CVaR_90
frontier_data = []
weights_data = []

for tr in target_returns:
    try:
        w, cvar, ret, _ = min_cvar_portfolio(
            returns=stock_returns_subset,
            alpha=a1,
            target_return=tr
        )
        _, cvar_a2 =
↳calculate_portfolio_return_and_cvar(returns_df=stock_returns_subset,
↳weights_array=w, alpha=0.9)
        frontier_data.append({'Target_Return': tr, 'Return': ret, 'CVaRa1':
↳cvar, 'CVaRa2': cvar_a2})
        weights_data.append(w)
    except Exception as e:
        print('infeasible return')
        continue

frontier_df1 = pd.DataFrame(frontier_data)
weights_df1 = pd.DataFrame(weights_data, index=target_returns,
↳columns=stock_returns_subset.columns)

#mean-CVaR_95
frontier_data = []
weights_data = []

for tr in target_returns:
    try:
        w, cvar, ret, _ = min_cvar_portfolio(
            returns=stock_returns_subset,
            alpha=a2,
            target_return=tr
        )

```

```

        _, cvar_a1 = □
    ↪calculate_portfolio_return_and_cvar(returns_df=stock_returns_subset, □
    ↪weights_array=w, alpha=a1)
        frontier_data.append({'Target_Return': tr, 'Return': ret, 'CVaRa2': □
    ↪cvar, 'CVaRa1': cvar_a1})
        weights_data.append(w)
    except Exception as e:
        print('infeasible return')
        continue

frontier_df2 = pd.DataFrame(frontier_data)
weights_df2 = pd.DataFrame(weights_data, index=target_returns, □
    ↪columns=stock_returns_subset.columns)

#CVaR_90-CVaR_95
frontier_data = []
weights_data = []

w, cvar, _, _ = min_cvar_portfolio(
    returns=stock_returns_subset,
    alpha=a1,
)
_, cvar2_max = □
    ↪calculate_portfolio_return_and_cvar(returns_df=stock_returns_subset, □
    ↪weights_array=w, alpha=a2)

_, cvar2_min, _, _ = min_cvar_portfolio(
    returns=stock_returns_subset,
    alpha=a2,
)

for tc2 in np.linspace(cvar2_min, cvar2_max, 100):
    try:
        w, cvar, ret, cvar2 = min_cvar_portfolio(
            returns=stock_returns_subset,
            alpha=a1,
            alpha2=a2,
            target_cvar2 = tc2
        )
        frontier_data.append({'Target_Return': tr, 'Return': ret, 'CVaRa1': □
    ↪cvar, 'CVaRa2': cvar2})
        weights_data.append(w)
    except Exception as e:
        print('infeasible return')
        continue

frontier_df3 = pd.DataFrame(frontier_data)

```

```
[48]: def plot_gradient_line(ax, x, y, c, cmap='viridis', norm=None, linewidth=2):
      """
      Plot a line colored by third variable c using LineCollection
      """
      points = np.array([x, y]).T.reshape(-1, 1, 2)
      segments = np.concatenate([points[:-1], points[1:]], axis=1)
      lc = LineCollection(segments, array=c, cmap=cmap, norm=norm,
                          linewidth=linewidth)
      ax.add_collection(lc)
      return lc
```

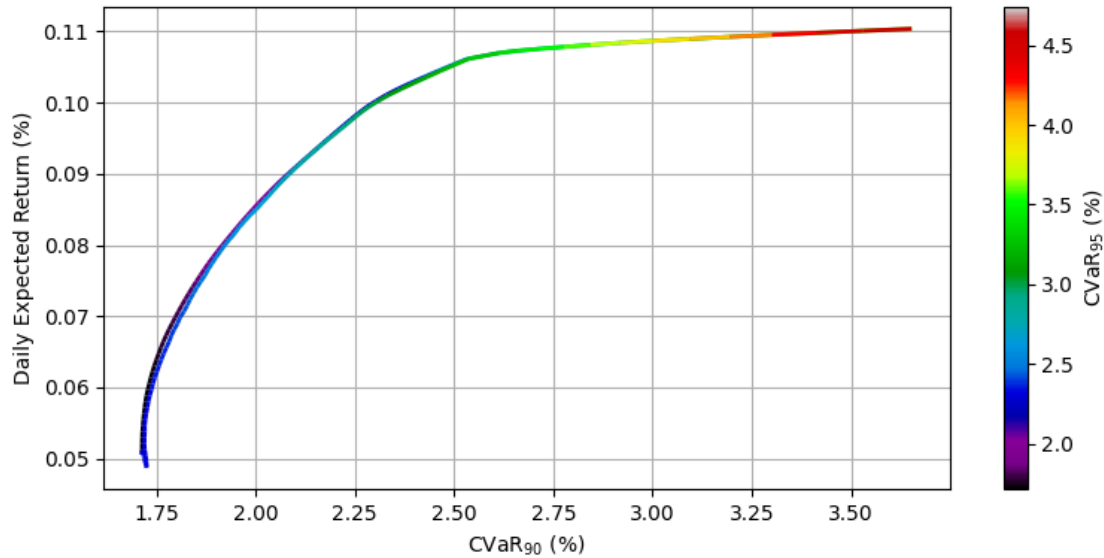
```
[49]: #plot of the efficient set for M3
fig, ax = plt.subplots(figsize=(8, 4))

# Normalize for colormap
all_c2 = np.concatenate([frontier_df1['CVaRa2'].values,
                          frontier_df2['CVaRa2'].values,
                          frontier_df3['CVaRa2'].values])
norm = plt.Normalize(vmin=all_c2.min(), vmax=all_c2.max())
cmap = cm.viridis

# Plot each frontier with gradient based on CVaRa2
for df in [(frontier_df1),
            (frontier_df2),
            (frontier_df3)]:
    x = df['CVaRa1'].values
    y = df['Return'].values
    c = df['CVaRa2'].values
    lc = plot_gradient_line(ax, x, y, c, cmap='nipy_spectral', norm=norm,
                            linewidth=2)
    ax.plot(x, y, color='none', label=label)

cbar = fig.colorbar(lc, ax=ax)
cbar.set_label('$\mathregular{CVaR}_{95}$ (%)')

ax.set_xlabel('$\mathregular{CVaR}_{90}$ (%)')
ax.set_ylabel('Daily Expected Return (%)')
ax.grid(True)
plt.tight_layout()
plt.savefig("CVaR_95_boundary.pdf", dpi=300, bbox_inches='tight',
            transparent=True)
plt.show()
```



[50]: *#Non-convex efficient set example from the thesis*

```

tr_1 = 0.06
tr_2 = 0.105
tr_3 = 0.085
a1=0.95
a2=0.9
tc2_1 = 1.732
tc2_2 = 2.484
tc2_3 = 2

w_1, cvar_1, ret_1, cvar2_1 = min_cvar_portfolio(
    returns=stock_returns_subset,
    alpha=a1,
    alpha2=a2,
    target_cvar2 = tc2_1,
    target_return=tr_1
)
w_2, cvar_2, ret_2, cvar2_2 = min_cvar_portfolio(
    returns=stock_returns_subset,
    alpha=a1,
    alpha2=a2,
    target_cvar2 = tc2_2,
    target_return=tr_2
)
w_3, cvar_3, ret_3, cvar2_3 = min_cvar_portfolio(
    returns=stock_returns_subset,
    alpha=a1,
    alpha2=a2,

```

```

        target_cvar2 = tc2_3,
        target_return=tr_3
    )
_, comb_cvar2=calculate_portfolio_return_and_cvar(stock_returns_subset, 0.
↳444*w_1+0.556*w_2, a2)
_, comb_cvar=calculate_portfolio_return_and_cvar(stock_returns_subset, 0.
↳444*w_1+0.556*w_2, a1)
print('Return, CVaR90 and CVaR95 for w_1:', ret_1, cvar2_1, cvar_1)
print('Return, CVaR90 and CVaR95 for w_2:', ret_2, cvar2_2, cvar_2)
print('CVaR90 and CVaR95 for 0.444w_1+0.556w_2:', comb_cvar2, comb_cvar)
print('Return, CVaR90 and CVaR95 for w_3:', ret_3, cvar2_3, cvar_3)

```

```

Return, CVaR90 and CVaR95 for w_1: 0.059999999999999514 1.7319999999767233
2.350693276287783
Return, CVaR90 and CVaR95 for w_2: 0.10499999999899695 2.483999999862909
3.2440017276457582
CVaR90 and CVaR95 for 0.444w_1+0.556w_2: 2.034741729542134 2.7270046688643528
Return, CVaR90 and CVaR95 for w_3: 0.08499999999999935 1.999999999530818
2.6496695215201447

```

```

[51]: # sets of portfolios for comparison, stored as 'M3_result_df'
target_returns = [0.06, 0.085, 0.105]
cvar1_level=0.9
cvar2_level=0.95
n = 50

records = []

for tr in target_returns:
    _, min_cvar, _, _ = min_cvar_portfolio(
        returns=stock_returns_subset,
        alpha=cvar1_level,
        target_return=tr
    )
    _, _, _, max_cvar = min_cvar_portfolio(
        returns=stock_returns_subset,
        alpha=cvar2_level,
        target_return=tr,
        alpha2=cvar1_level,
        target_cvar2=999
    )

    cvar2_values = np.linspace(min_cvar, max_cvar, n)
    for tc2 in cvar2_values:
        try:
            w, cvar1, ret, cvar2 = min_cvar_portfolio(
                returns=stock_returns_subset,

```

```

        alpha=cvar2_level,
        target_return=tr,
        alpha2=cvar1_level,
        target_cvar2=tc2
    )
    port_rets = stock_returns_subset @ w
    cum = (1 + port_rets/100).cumprod()
    single_day_max_loss = -port_rets.values.min()
    max_dd = max_drawdown(cum.values)*100
    records.append({'Target_Return': tr, 'Target_CVaR': tc2, 'Return':␣
↪ret, 'CVaR': cvar2, 'MaxDailyLoss': single_day_max_loss, 'MaxDrawdown':␣
↪max_dd, 'Weights': w})
    except Exception:
        continue

M3_result_df = pd.DataFrame(records)

```

```

[52]: # Data preparation for the weights plot of different CVaR trade-offs at␣
↪expected return 0.085%
tr=0.085
cvar1_level=0.9
cvar2_level=0.95
records = []
counter = 0

_, min_cvar, _, _ = min_cvar_portfolio(
    returns=stock_returns_subset,
    alpha=cvar1_level,
    target_return=tr
)
_, _, _, max_cvar = min_cvar_portfolio(
    returns=stock_returns_subset,
    alpha=cvar2_level,
    target_return=tr,
    alpha2=cvar1_level,
    target_cvar2=999
)

cvars = np.linspace(min_cvar, max_cvar, 50)
for tc2 in cvars:
    try:
        w, cvar1, ret, cvar2 = min_cvar_portfolio(
            returns=stock_returns_subset,
            alpha=cvar2_level,
            target_return=tr,
            alpha2=cvar1_level,

```

```

        target_cvar2=tc2
    )
    records.append({'Target_Return': tr, 'Target_CVaR2': tc2, 'Return':
↳ret, 'CVaR1': cvar1, 'CVaR2': cvar2, 'Weights': w})
    except Exception:
        counter = counter+1
        continue

df_temp = pd.DataFrame(records)

weights_expanded = pd.DataFrame(df_temp['Weights'].to_list(),
                                index=df_temp['Target_CVaR2'],
                                columns=stock_returns_subset.columns)

```

```

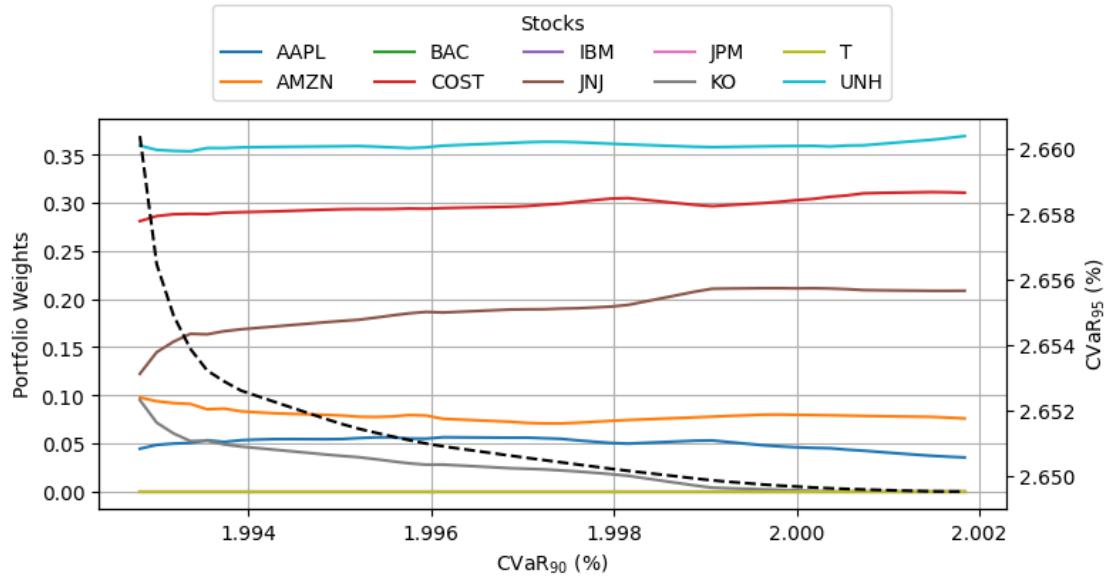
[53]: fig, ax = plt.subplots(figsize=(8, 4))

weights_expanded.plot(ax=ax,
                      linestyle='-',
                      alpha=1.0,
                      linewidth=1.5)

ax.set_xlabel('$\mathregular{CVaR}_{90}$ (%)')
ax.set_ylabel('Portfolio Weights')
ax.grid(True)
ax.legend(title="Stocks", ncol=1)
ax2 = ax.twinx()
ax2.plot(df_temp['Target_CVaR2'], df_temp['CVaR1'], color='black',
         linestyle='--')
ax2.set_ylabel('$\mathregular{CVaR}_{95}$ (%)')
box = ax.get_position()
ax.set_position([box.x0, box.y0 + 0.1, box.width, box.height * 0.9])

ax.legend(loc='lower center', bbox_to_anchor=(0.5, 1.02),
         ncol=5, title="Stocks")
plt.tight_layout()
plt.savefig("CVaR_95_weights.pdf", dpi=300, bbox_inches='tight',
↳transparent=True)
plt.show()

```



2.0.5 Model M4

This model considers CVaR₉₀ and CVaR_{99.5}.

```
[54]: # Data preparation for the weights plot of different CVaR trade-offs at
      ↪ expected return 0.085%
tr=0.085
cvar1_level=0.9
cvar2_level=0.995
records = []
counter = 0

_, min_cvar, _, _ = min_cvar_portfolio(
    returns=stock_returns_subset,
    alpha=cvar1_level,
    target_return=tr
)
print(min_cvar)
_, _, _, max_cvar = min_cvar_portfolio(
    returns=stock_returns_subset,
    alpha=cvar2_level,
    target_return=tr,
    alpha2=cvar1_level,
    target_cvar2=999
)
print(max_cvar)
```

```

cvars = np.linspace(min_cvar, max_cvar, 50)
for tc2 in cvars:
    try:
        w, cvar1, ret, cvar2 = min_cvar_portfolio(
            returns=stock_returns_subset,
            alpha=cvar2_level,
            target_return=tr,
            alpha2=cvar1_level,
            target_cvar2=tc2
        )
        records.append({'Target_Return': tr, 'Target_CVaR2': tc2, 'Return': ret,
            'CVaR1': cvar1, 'CVaR2': cvar2, 'Weights': w})
    except Exception:
        counter = counter+1
        continue

df_temp = pd.DataFrame(records)

weights_expanded = pd.DataFrame(df_temp['Weights'].to_list(),
                                index=df_temp['Target_CVaR2'],
                                columns=stock_returns_subset.columns)

```

```

1.992813250057119
2.234058035749467

```

```

[55]: fig, ax = plt.subplots(figsize=(8, 4))

weights_expanded.plot(ax=ax,
                      linestyle='-',
                      alpha=1.0,
                      linewidth=1.5)

ax.set_xlabel('$\mathregular{CVaR}_{90}$ (%)')
ax.set_ylabel('Portfolio Weights')
ax.grid(True)
ax.legend(title="Stocks", ncol=1)
ax2 = ax.twinx()
ax2.plot(df_temp['Target_CVaR2'], df_temp['CVaR1'], color='black',
         linestyle='--')
ax2.set_ylabel('$\mathregular{CVaR}_{99.5}$ (%)')

cv_lines = [1.993, 2.022]
labels = ['M1', 'M2']

ymin, ymax = ax.get_ylim()
y_text = ymax * 0.98

```

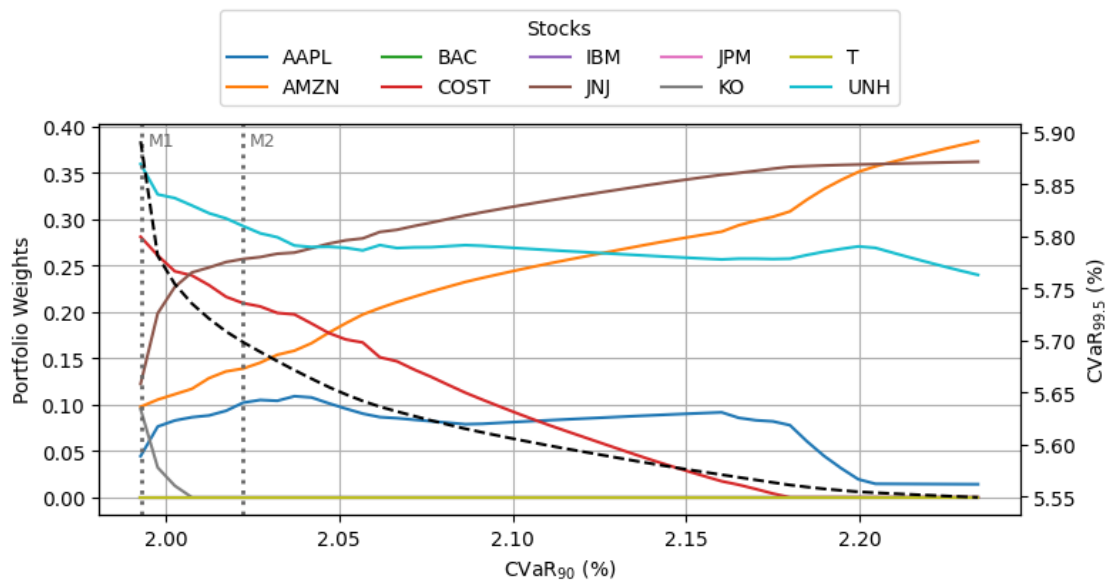
```

for x_val, lbl in zip(cv_lines, labels):
    ax.axvline(x=x_val, color='dimgray', linestyle=':', linewidth=2)
    ax.text(x_val+0.002, y_text, lbl,
            color='dimgray', fontsize=9,
            ha='left', va='top')

box = ax.get_position()
ax.set_position([box.x0, box.y0 + 0.1, box.width, box.height * 0.9])

ax.legend(loc='lower center', bbox_to_anchor=(0.5, 1.02),
          ncol=5, title="Stocks")
plt.tight_layout()
plt.savefig("CVaR_995_weights.pdf", dpi=300, bbox_inches='tight',
           transparent=True)
plt.show()

```



```

[56]: # calculate portfolios in the efficient set of  $M_4$ , WARNING, longer run time.
n=100
m=30
cvar1_level=0.9
cvar2_level=0.995
cvar2_values = []
for col in stock_returns_subset.columns:
    cvar = asset_cvar(stock_returns_subset[col], cvar2_level)
    cvar2_values.append(cvar)

```

```

base = np.linspace(stock_returns_subset.mean().min(), stock_returns_subset.
↳mean().max(), n)
top5 = np.sort(base)[-5:] # pick top 5 highest
extra_ranges = [np.linspace(top5[i], top5[i+1], 3, endpoint=False)[1:]
↳for i in range(4)]
extra = np.concatenate(extra_ranges)
target_returns = np.sort(np.unique(np.concatenate([base, extra]))) #increased↳
↳density in the highest target return, where portfolios significantly differ

records = []
counter = 0

for tr in tqdm(target_returns, desc="returns progress"):
    w_temp, _, _, _ = min_cvar_portfolio(
        returns=stock_returns_subset,
        alpha=cvar1_level,
        target_return=tr
    )
    _, cvar2_temp_max =↳
↳calculate_portfolio_return_and_cvar(returns_df=stock_returns_subset,↳
↳weights_array=w_temp, alpha=cvar2_level)
    _, cvar2_temp_min, _, _ = min_cvar_portfolio(
        returns=stock_returns_subset,
        alpha=cvar2_level,
        target_return=tr
    )
    u = np.linspace(0, 1, m)
    skewed_returns = cvar2_temp_min + (cvar2_temp_max - cvar2_temp_min) * (u**2)
    for tc2 in skewed_returns:
        try:
            w, cvar1, ret, cvar2 = min_cvar_portfolio(
                returns=stock_returns_subset,
                alpha=cvar1_level,
                target_return=tr,
                alpha2=cvar2_level,
                target_cvar2=tc2
            )
            records.append({'Target_Return': tr, 'Target_CVaR2': tc2, 'Return':↳
↳ret, 'CVaR1': cvar1, 'CVaR2': cvar2, 'Weights': w})
        except Exception:
            counter = counter+1
            continue

df2 = pd.DataFrame(records)

```

returns progress:

100%|

| 108/108

[08:19<00:00, 4.62s/it]

```
[57]: # uncomment to save the calculated set of portfolios to avoid repeated long_
      ↪ calculations
      # df2.to_csv('efficient_cvar_frontier.csv', index=False)
      # print("DataFrame saved to efficient_cvar_frontier.csv")
```

```
[58]: #find the "boundary" of efficient set
base = np.linspace(stock_returns_subset.mean().min(), stock_returns_subset.
      ↪ mean().max(), n)
top5 = np.sort(base)[-5:] # pick top 5 highest
extra_ranges = [np.linspace(top5[i], top5[i+1], 3, endpoint=False)[1:]
      ↪ for i in range(4)]
extra = np.concatenate(extra_ranges)
target_returns = np.sort(np.unique(np.concatenate([base, extra])))
a1=0.9
a2=0.995

#mean-CVaR_90
frontier_data = []
weights_data = []

for tr in target_returns:
    try:
        w, cvar, ret, _ = min_cvar_portfolio(
            returns=stock_returns_subset,
            alpha=a1,
            target_return=tr
        )
        _, cvar_a2 =
      ↪ calculate_portfolio_return_and_cvar(returns_df=stock_returns_subset,
      ↪ weights_array=w, alpha=0.9)
        frontier_data.append({'Target_Return': tr, 'Return': ret, 'CVaRa1':
      ↪ cvar, 'CVaRa2': cvar_a2})
        weights_data.append(w)
    except Exception as e:
        print('infeasible return')
        continue

frontier_df1 = pd.DataFrame(frontier_data)
weights_df1 = pd.DataFrame(weights_data, index=target_returns,
      ↪ columns=stock_returns_subset.columns)

#mean-CVaR_99.5
frontier_data = []
weights_data = []
```

```

for tr in target_returns:
    try:
        w, cvar, ret, _ = min_cvar_portfolio(
            returns=stock_returns_subset,
            alpha=a2,
            target_return=tr
        )
        _, cvar_a1 =
↳calculate_portfolio_return_and_cvar(returns_df=stock_returns_subset,
↳weights_array=w, alpha=a1)
        frontier_data.append({'Target_Return': tr, 'Return': ret, 'CVaRa2':
↳cvar, 'CVaRa1': cvar_a1})
        weights_data.append(w)
    except Exception as e:
        print('infeasible return')
        continue

frontier_df2 = pd.DataFrame(frontier_data)
weights_df2 = pd.DataFrame(weights_data, index=target_returns,
↳columns=stock_returns_subset.columns)

#CVaR_90-CVaR_99.5
frontier_data = []
weights_data = []

w, cvar, _, _ = min_cvar_portfolio(
    returns=stock_returns_subset,
    alpha=a1,
)
_, cvar2_max =
↳calculate_portfolio_return_and_cvar(returns_df=stock_returns_subset,
↳weights_array=w, alpha=a2)

_, cvar2_min, _, _ = min_cvar_portfolio(
    returns=stock_returns_subset,
    alpha=a2,
)

for tc2 in np.linspace(cvar2_min, cvar2_max, 100):
    try:
        w, cvar, ret, cvar2 = min_cvar_portfolio(
            returns=stock_returns_subset,
            alpha=a1,
            alpha2=a2,
            target_cvar2 = tc2
        )

```

```

        frontier_data.append({'Target_Return': tr, 'Return': ret, 'CVaRa1': cvar,
        ↪cvar, 'CVaRa2': cvar2})
        weights_data.append(w)
    except Exception as e:
        print('infeasible return')
        continue

frontier_df3 = pd.DataFrame(frontier_data)

```

```

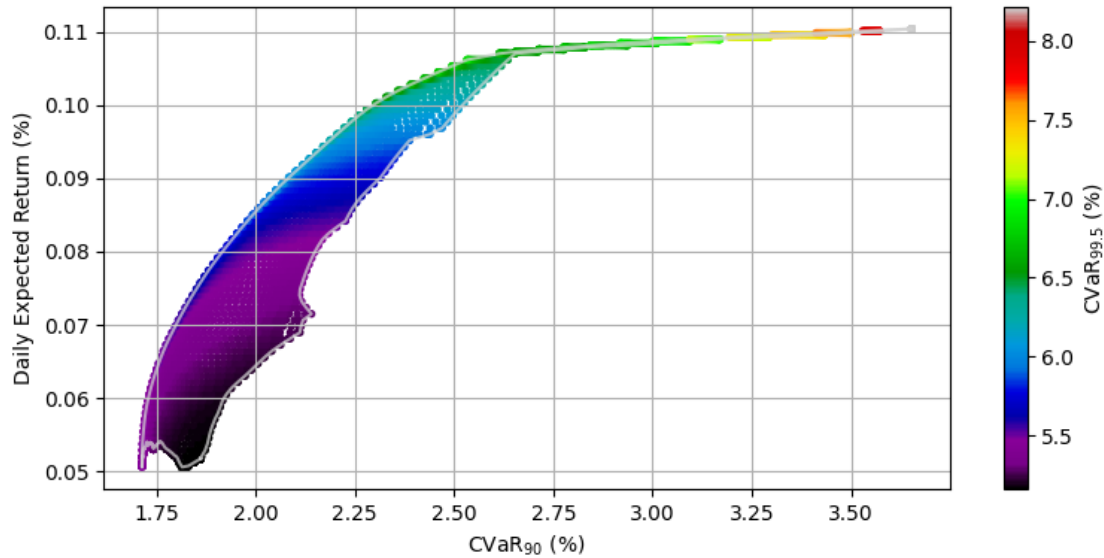
[59]: plt.figure(figsize=(8, 4))
scatter = plt.scatter(
    df2['CVaR1'],
    df2['Return'],
    c=df2['CVaR2'],
    cmap='nipy_spectral',
    marker='o',
    s=8,
)
plt.plot(frontier_df1['CVaRa1'], frontier_df1['Return'], markersize=4,
↪color='lightgray', alpha=0.75)
plt.plot(frontier_df2['CVaRa1'], frontier_df2['Return'], markersize=4,
↪color='lightgray', alpha=0.75)
plt.plot(frontier_df3['CVaRa1'], frontier_df3['Return'], markersize=4,
↪color='lightgray', alpha=0.75)

cbar = plt.colorbar(scatter)
cbar.set_label('$\mathregular{CVaR_{99.5}}$ (%)')

plt.xlabel('$\mathregular{CVaR_{90}}$ (%)')
plt.ylabel('Daily Expected Return (%)')
plt.grid(True)

plt.tight_layout()
plt.savefig("CVaR_995_set.pdf", dpi=300, bbox_inches='tight', transparent=True)
plt.show()

```



```
[60]: # sets of portfolios for comparison, stored as 'M4_result_df'
target_returns = [0.06, 0.085, 0.105]
cvar1_level=0.9
cvar2_level=0.995
n = 50

records = []

for tr in target_returns:
    _, min_cvar, _, _ = min_cvar_portfolio(
        returns=stock_returns_subset,
        alpha=cvar1_level,
        target_return=tr
    )
    _, _, _, max_cvar = min_cvar_portfolio(
        returns=stock_returns_subset,
        alpha=cvar2_level,
        target_return=tr,
        alpha2=cvar1_level,
        target_cvar2=999
    )
    print(min_cvar, max_cvar)
    cvar2_values = np.linspace(min_cvar, max_cvar, n)
    for tc2 in cvar2_values:
        try:
            w, cvar1, ret, cvar2 = min_cvar_portfolio(
                returns=stock_returns_subset,
                alpha=cvar2_level,
```

```

        target_return=tr,
        alpha2=cvar1_level,
        target_cvar2=tc2
    )
    port_rets = stock_returns_subset @ w
    cum = (1 + port_rets/100).cumprod()
    single_day_max_loss = -port_rets.values.min()
    max_dd = max_drawdown(cum.values)*100
    records.append({'Target_Return': tr, 'Target_CVaR': tc2, 'Return':␣
↳ret, 'CVaR': cvar2, 'MaxDailyLoss': single_day_max_loss, 'MaxDrawdown':␣
↳max_dd, 'Weights': w})
    except Exception:
        continue

M4_result_df = pd.DataFrame(records)

```

```

1.729880686814393 1.9198759049229248
1.992813250057119 2.234058035749467
2.482180205399854 2.613051475650358

```

```

[61]: # Example of non-convexity of the efficient set
tr_1 = 0.06
tr_2 = 0.105
tr_3 = 0.085
a1=0.995
a2=0.9
tc2_1 = 1.8
tc2_2 = 2.5
tc2_3 = 2.05

w_1, cvar_1, ret_1, cvar2_1 = min_cvar_portfolio(
    returns=stock_returns_subset,
    alpha=a1,
    alpha2=a2,
    target_cvar2 = tc2_1,
    target_return=tr_1
)
w_2, cvar_2, ret_2, cvar2_2 = min_cvar_portfolio(
    returns=stock_returns_subset,
    alpha=a1,
    alpha2=a2,
    target_cvar2 = tc2_2,
    target_return=tr_2
)
w_3, cvar_3, ret_3, cvar2_3 = min_cvar_portfolio(
    returns=stock_returns_subset,
    alpha=a1,

```

```

        alpha2=a2,
        target_cvar2 = tc2_3,
        target_return=tr_3
    )
_, comb_cvar2=calculate_portfolio_return_and_cvar(stock_returns_subset, 0.
↳444*w_1+0.556*w_2, a2)
_, comb_cvar=calculate_portfolio_return_and_cvar(stock_returns_subset, 0.
↳444*w_1+0.556*w_2, a1)
print('Return, CVaR90 and CVaR99.5 for w_1:', ret_1, cvar2_1, cvar_1)
print('Return, CVaR90 and CVaR99.5 for w_2:', ret_2, cvar2_2, cvar_2)
print('CVaR90 and CVaR99.5 for 0.444w_1+0.556w_2:', comb_cvar2, comb_cvar)
print('Return, CVaR90 and CVaR99.5 for w_3:', ret_3, cvar2_3, cvar_3)

```

```

Return, CVaR90 and CVaR99.5 for w_1: 0.05999999999993115 1.7999999977268981
5.260192987709949
Return, CVaR90 and CVaR99.5 for w_2: 0.105000000000003697 2.49999999979242
6.566928447901884
CVaR90 and CVaR99.5 for 0.444w_1+0.556w_2: 2.079667831354419 5.729844161844336
Return, CVaR90 and CVaR99.5 for w_3: 0.08499999999999655 2.049999999527807
5.651034946048732

```

```

[62]: # data preparation for plot with contour lines
cvar2_constraints = [5.2, 5.3, 5.4, 5.6, 5.9]
frontiers = {}

n = 250
target_returns = np.linspace(stock_returns_subset.mean().min(),
↳stock_returns_subset.mean().max(), n)

for constraint in cvar2_constraints:
    records = []
    for tr in target_returns:
        try:
            w, cvar, ret, cvar2 = min_cvar_portfolio(
                returns=stock_returns_subset,
                alpha=0.9,
                target_return=tr,
                alpha2=0.995,
                target_cvar2=constraint if constraint is not None else 1e6 #↳
↳effectively unconstrained
            )
            records.append({'Target_Return': tr, 'Return': ret, 'CVaR': cvar,
↳'CVaR2': cvar2, 'Weights': w})
        except Exception:
            continue

# Create DataFrame for this constraint level

```

```

    frontier_df = pd.DataFrame([
        {'Target_Return': rec['Target_Return'], 'Return': rec['Return'], 'CVaR':
↪ rec['CVaR'], 'CVaR2': rec['CVaR2']}
        for rec in records
    ])
    frontiers[constraint] = frontier_df

cvar2_constraints = [6.2]
target_returns = np.linspace(stock_returns_subset.mean().min(),
↪ stock_returns_subset.mean(), 2*n)
for constraint in cvar2_constraints:
    records = []
    for tr in target_returns:
        try:
            w, cvar, ret, cvar2 = min_cvar_portfolio(
                returns=stock_returns_subset,
                alpha=0.9,
                target_return=tr,
                alpha2=0.995,
                target_cvar2=constraint if constraint is not None else 1e6 #
↪ effectively unconstrained
            )
            records.append({'Target_Return': tr, 'Return': ret, 'CVaR': cvar,
↪ 'CVaR2': cvar2, 'Weights': w})
        except Exception:
            continue

    # Create DataFrame for this constraint level
    frontier_df = pd.DataFrame([
        {'Target_Return': rec['Target_Return'], 'Return': rec['Return'], 'CVaR':
↪ rec['CVaR'], 'CVaR2': rec['CVaR2']}
        for rec in records
    ])
    frontiers[constraint] = frontier_df

```

```

[63]: #plot with contour lines
fig, ax = plt.subplots(figsize=(8, 4))

scatter = ax.scatter(
    df2['CVaR1'], df2['Return'],
    c=df2['CVaR2'], cmap='nipy_spectral', s=8, alpha=0.25
)
plt.colorbar(scatter, label='$\\mathregular{CVaR_{99.5}}$ (%)')

ax.plot(frontier_df1['CVaRa1'], frontier_df1['Return'], color='lightgray',
↪ alpha=0.75)

```

```

ax.plot(frontier_df2['CVaRa1'], frontier_df2['Return'], color='lightgray',
        ↪alpha=0.75)
ax.plot(frontier_df3['CVaRa1'], frontier_df3['Return'], color='lightgray',
        ↪alpha=0.75)

for constraint, df in frontiers.items():
    if constraint is None:
        continue

    # Filter rows where CVaR2 >= constraint - 0.01 (to filter out parts of
    ↪contours where achieved CVaR_99.5 is lower than the constraint
    threshold = constraint - 0.01
    df_trim = df[df['CVaR2'] >= threshold]
    if df_trim.empty:
        continue

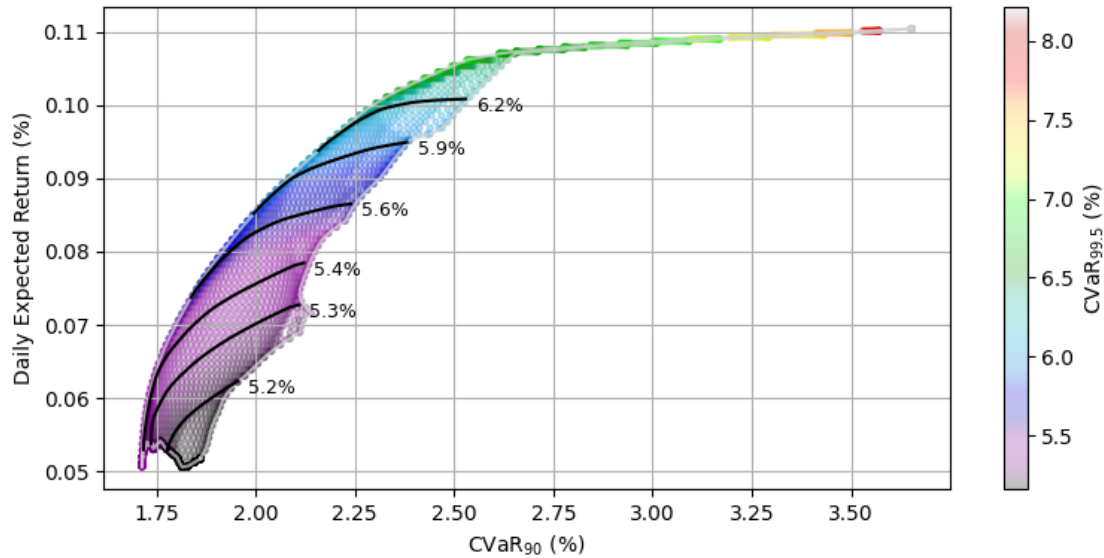
    x = df_trim['CVaR'].values
    y = df_trim['Return'].values

    ax.plot(x, y, color='black', linewidth=1.5)

    ax.annotate(
        f"{constraint:.1f}%",
        xy=(x[-1], y[-1]-0.001),
        xytext=(5, 0),
        textcoords='offset points',
        ha='left', va='center',
        fontsize=9, color='black'
    )

ax.set_xlabel('$\mathregular{CVaR_{90}}$ (%)')
ax.set_ylabel('Daily Expected Return (%)')
ax.grid(True)
plt.tight_layout()
plt.savefig("CVaR_995_contours.pdf", dpi=300, bbox_inches='tight',
        ↪transparent=True)
plt.show()

```



2.0.6 Transaction cost model (M5)

This model will require new minimization functions that can consider L1 distance of portfolios.

```
[64]: # similar to CVaR minimization before with additional arguments for max L1
      ↪ distance and starting portfolio
def min_cvar_w_cost(returns: pd.DataFrame, alpha: float, max_l1: float, w0: np.
      ↪ ndarray, target_return: float | None = None):
    R = returns.values # shape (T, n)
    T, n = R.shape

    # Variables
    w = cp.Variable(n)
    z = cp.Variable()
    u = cp.Variable(T)

    # Objective: z + (1 / (T*(1-alpha))) * sum(u)
    obj = cp.Minimize(z + cp.sum(u) / (T * (1 - alpha)))

    # Constraints:
    constraints = [
        cp.sum(w) == 1,
        w >= 0,
        u >= 0,
        u >= -(R @ w) - z
    ]

    if target_return is not None:
```

```

    mu = returns.mean().values
    constraints.append(w @ mu >= target_return)

constraints.append(cp.norm(w - w0, 1) <= max_l1)

prob = cp.Problem(obj, constraints)
prob.solve(solver=cp.ECOS)

# Results
weights = w.value
losses = -R @ weights
var_threshold = np.quantile(losses, alpha)
optimal_cvar = var_threshold + np.mean(np.maximum(losses - var_threshold,
↳0)) / (1 - alpha)
distance = np.linalg.norm(weights - w0, ord=1)
exp_return = np.dot(returns.mean(), weights)

return weights, optimal_cvar, exp_return, distance

```

```

[65]: # to set limits when finding the entire efficient set, we will need max_
↳achievable return at given transaction cost
def max_return_w_cost(returns: pd.DataFrame, max_l1: float, w0: np.ndarray,
↳alpha: float | None = None, target_cvar: float | None = None):

    R = returns.values
    T, n = R.shape
    mu = returns.mean().values

    # Variables
    w = cp.Variable(n)

    obj = cp.Minimize(- w @ mu)

    # Constraints:
    constraints = [
        cp.sum(w) == 1,          # fully invested
        w >= 0,                 # long-only
    ]

    constraints.append(cp.norm(w - w0, 1) <= max_l1)

    if (alpha is None) ^ (target_cvar is None):
        raise ValueError("Specify both alpha2 and target_cvar2 together")
    if alpha is not None:

```

```

z2 = cp.Variable()
u2 = cp.Variable(T)
constraints += [
    u2 >= 0,
    u2 >= -(R @ w) - z2
]
cvar = z2 + cp.sum(u2) / (T * (1 - alpha))
constraints.append(cvar <= target_cvar)

# Solve
prob = cp.Problem(obj, constraints)
prob.solve(solver=cp.ECOS)

# Results
weights = w.value
losses = -R @ weights
distance = np.linalg.norm(weights - w0, ord=1)
exp_return = np.dot(returns.mean(), weights)
cvar=0
if alpha is not None:
    var_threshold = np.quantile(losses, alpha)
    cvar = var_threshold + np.mean(np.maximum(losses - var_threshold, 0)) /
↳(1 - alpha)

return weights, exp_return, distance, cvar

```

```

[81]: # finds portfolios in efficient set of model (M5) WARNING, longer run for high
↳m, n
n=100
m=50
cvar1_level=0.9
k=stock_returns_subset.shape[1]
start_port=np.ones(k) / k
#start_port[2]=1

distances = np.linspace(0, 2, n)

records = []
counter = 0

for dist in tqdm(distances, desc="returns progress"):
    _, _, min_ret, _ = min_cvar_w_cost(
        returns=stock_returns_subset,
        alpha=cvar1_level,
        max_l1=dist,
        w0 = start_port
    )

```

```

    _, max_ret, _, _ = max_return_w_cost(returns=stock_returns_subset,
↪max_l1=dist, w0=start_port)
    base = np.linspace(min_ret, max_ret, m)
    top5 = np.sort(base)[-4:] # pick top 4 highest
    extra_ranges = [np.linspace(top5[i], top5[i+1], 5, endpoint=False)[1:]
                    for i in range(3)]
    extra = np.concatenate(extra_ranges)
    target_returns = np.sort(np.unique(np.concatenate([base, extra])))
    for tr in target_returns:
        try:
            w, cvar1, ret, dis = min_cvar_w_cost(
                returns=stock_returns_subset,
                alpha=cvar1_level,
                max_l1=dist,
                w0 = start_port,
                target_return=tr
            )
            records.append({'Target_Return': tr, 'Target_Distance': dist,
↪'Return': ret, 'CVaR1': cvar1, 'Distance': dis, 'Weights': w})
        except Exception:
            counter = counter+1
            continue

df_cost = pd.DataFrame(records)

```

```

returns progress:
100%|                                     | 100/100
[04:15<00:00, 2.56s/it]

```

```

[67]: #find the "boundary" of efficient set
n=100
target_returns = np.linspace(stock_returns_subset.mean().min(),
↪stock_returns_subset.mean().max(), n)
target_distances = np.linspace(0, 2, n)
a1=0.9

#mean-CVaR_90
frontier_data = []
weights_data = []

for tr in target_returns:
    try:
        w, cvar, ret, _ = min_cvar_portfolio(
            returns=stock_returns_subset,
            alpha=a1,
            target_return=tr
        )

```

```

        dis = l1_distance(w, start_port)
        frontier_data.append({'Target_Return': tr, 'Return': ret, 'CVaR': cvar,
↪'Distance': dis})
        weights_data.append(w)
    except Exception as e:
        print('infeasible return')
        continue

frontier_df1 = pd.DataFrame(frontier_data)
weights_df1 = pd.DataFrame(weights_data, index=target_returns,
↪columns=stock_returns_subset.columns)

frontier_data = []
weights_data = []

#cost-CVaR_90
for dist in target_distances:
    try:
        w, cvar, ret, dis = min_cvar_w_cost(
            returns=stock_returns_subset,
            alpha=a1,
            max_l1=dist,
            w0 = start_port
        )
        frontier_data.append({'Target_Distance': dist, 'Return': ret, 'CVaR':
↪cvar, 'Distance': dis})
        weights_data.append(w)
    except Exception as e:
        print('infeasible return')
        continue

frontier_df2 = pd.DataFrame(frontier_data)
weights_df2 = pd.DataFrame(weights_data, index=target_distances,
↪columns=stock_returns_subset.columns)

#mean-cost
frontier_data = []
weights_data = []

for dist in target_distances:
    try:
        w, ret, dis, _ = max_return_w_cost(returns=stock_returns_subset,
↪max_l1=dist, w0=start_port)
        _, cvar =
↪calculate_portfolio_return_and_cvar(returns_df=stock_returns_subset,
↪weights_array=w, alpha=a1)

```

```

        frontier_data.append({'Target_Distance': dist, 'Return': ret, 'CVaR':
↪cvar, 'Distance': dis})
        weights_data.append(w)
    except Exception as e:
        print('infeasible return')
        continue

frontier_df3 = pd.DataFrame(frontier_data)
weights_df3 = pd.DataFrame(weights_data, index=target_distances,
↪columns=stock_returns_subset.columns)

```

```

[82]: #plot the entire efficient set
plt.figure(figsize=(8, 4))
scatter = plt.scatter(
    df_cost['CVaR1'],
    df_cost['Return'],
    c=df_cost['Distance'],
    cmap='nipy_spectral',
    marker='o',
    s=15,
)

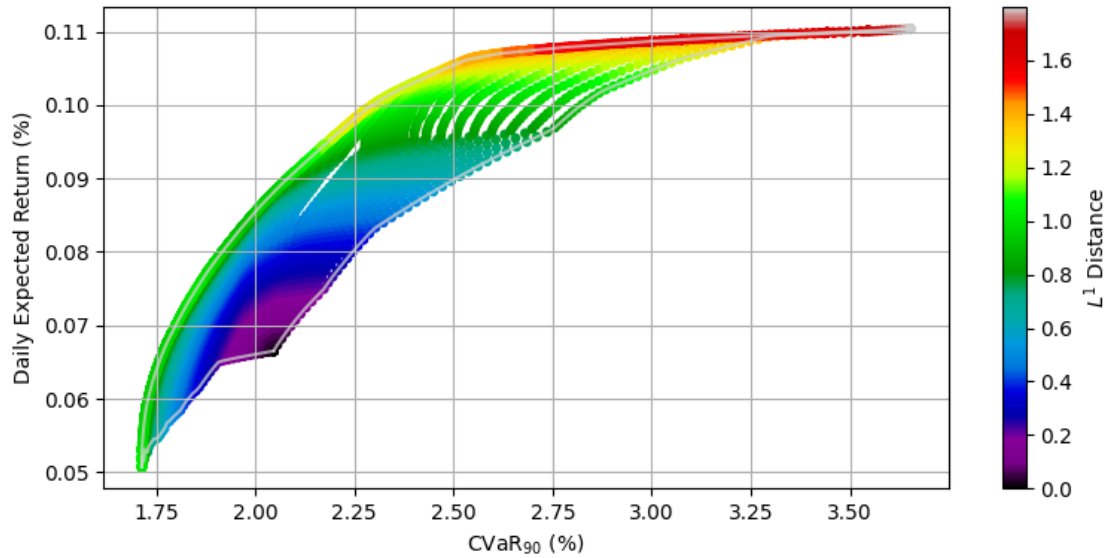
plt.plot(frontier_df1['CVaR'], frontier_df1['Return'], markersize=4,
↪color='lightgray', alpha=0.75)
plt.plot(frontier_df2['CVaR'], frontier_df2['Return'], markersize=4,
↪color='lightgray', alpha=0.75)
plt.plot(frontier_df3['CVaR'], frontier_df3['Return'], markersize=4,
↪color='lightgray', alpha=0.75)

cbar = plt.colorbar(scatter)
cbar.set_label('$L^1$ Distance')

plt.xlabel('$\mathregular{CVaR}_{90}$ (%)')
plt.ylabel('Daily Expected Return (%)')
plt.grid(True)

plt.tight_layout()
plt.savefig("cost_set.pdf", dpi=300, bbox_inches='tight', transparent=True)
plt.show()

```



```
[69]: # non-convexity of efficient set example
tr_1 = 0.06
tr_2 = 0.105
tr_3 = 0.085
a1=0.9
td_1 = 0.85
td_2 = 1.25
td_3 = 0.75
start_port=np.ones(10) / 10

w_1, cvar_1, ret_1, dis_1 = min_cvar_w_cost(
    returns=stock_returns_subset,
    alpha=a1,
    max_l1=td_1,
    w0 = start_port,
    target_return = tr_1
)
w_2, cvar_2, ret_2, dis_2 = min_cvar_w_cost(
    returns=stock_returns_subset,
    alpha=a1,
    max_l1=td_2,
    w0 = start_port,
    target_return = tr_2
)
w_3, cvar_3, ret_3, dis_3 = min_cvar_w_cost(
    returns=stock_returns_subset,
    alpha=a1,
    max_l1=td_3,
```

```

        w0 = start_port,
        target_return = tr_3
    )
comb_dist = l1_distance(start_port, 0.444*w_1+0.556*w_2)
_, comb_cvar=calculate_portfolio_return_and_cvar(stock_returns_subset, 0.
↳444*w_1+0.556*w_2, a1)
print('Return, CVaR90 and Distance for w_1:', ret_1, cvar2_1, dis_1)
print('Return, CVaR90 and Distance for w_2:', ret_2, cvar2_2, dis_2)
print('Distance and CVaR90 for 0.444w_1+0.556w_2:', comb_dist, comb_cvar)
print('Return, CVaR90 and Distance for w_3:', ret_3, cvar2_3, dis_3)

```

```

Return, CVaR90 and Distance for w_1: 0.05999999999999808 1.7999999977268981
0.84999999999919712
Return, CVaR90 and Distance for w_2: 0.10499999999999837 2.49999999979242
1.250000000000023
Distance and CVaR90 for 0.444w_1+0.556w_2: 0.780234858106249 2.067026496095681
Return, CVaR90 and Distance for w_3: 0.08499999999999813 2.049999999527807
0.7499999999995998

```

```

[70]: # prepare data for contour lines in a plot
cvar_constraints = [1.72, 1.75, 1.8, 1.85, 1.9, 2, 2.1, 2.2, 2.3, 2.5, 3, 3.5]
frontiers = {}

n = 100
target_distances = np.linspace(0, 2, n)

for constraint in cvar_constraints:
    records = []
    for dist in target_distances:
        try:
            w, ret, dis, cvar = max_return_w_cost(returns=stock_returns_subset,
↳max_l1=dist, w0=start_port, alpha=0.9, target_cvar=constraint)
            records.append({'Target_Distance': dist, 'Return': ret, 'CVaR':
↳cvar, 'Distance': dis, 'Weights': w})
        except Exception:
            continue

    # Create DataFrame for this constraint level
    frontier_df = pd.DataFrame([
        {'Target_Distance': rec['Target_Distance'], 'Return': rec['Return'],
↳'CVaR': rec['CVaR'], 'Distance': rec['Distance']}
        for rec in records
    ])
    frontiers[constraint] = frontier_df

```

```

[83]: # plot with conour line and L1 distance x axis
fig, ax = plt.subplots(figsize=(8, 4))

```

```

scatter = ax.scatter(
    df_cost['Distance'],
    df_cost['Return'],
    c=df_cost['CVaR1'],
    cmap='nipy_spectral',
    marker='o',
    s=8,
    #edgecolor='k',
    alpha=0.25
)
plt.colorbar(scatter, label='$\\mathregular{CVaR}_{90}$ (%)')

plt.plot(frontier_df1['Distance'], frontier_df1['Return'], markersize=4,
         ↪color='dimgray', alpha=0.75)
plt.plot(frontier_df2['Distance'], frontier_df2['Return'], markersize=4,
         ↪color='dimgray', alpha=0.75)
plt.plot(frontier_df3['Distance'], frontier_df3['Return'], markersize=4,
         ↪color='dimgray', alpha=0.75)

for constraint, df in frontiers.items():
    if constraint is None:
        continue

    # Filter rows where CVaR2 >= constraint - 0.01 (once again limiting
    ↪contours to not include lower CVaR values)
    threshold = constraint - 0.0001
    df_trim = df[df['CVaR'] >= threshold]
    if df_trim.empty:
        continue

    x = df_trim['Distance'].values
    y = df_trim['Return'].values

    ax.plot(x, y, color='black', linewidth=1.5)

    ax.annotate(
        f"{constraint:.2f}%",
        xy=(x[-1], y[-1]-0.001),
        xytext=(5, 0),
        textcoords='offset points',
        ha='left', va='center',
        fontsize=9, color='black'
    )

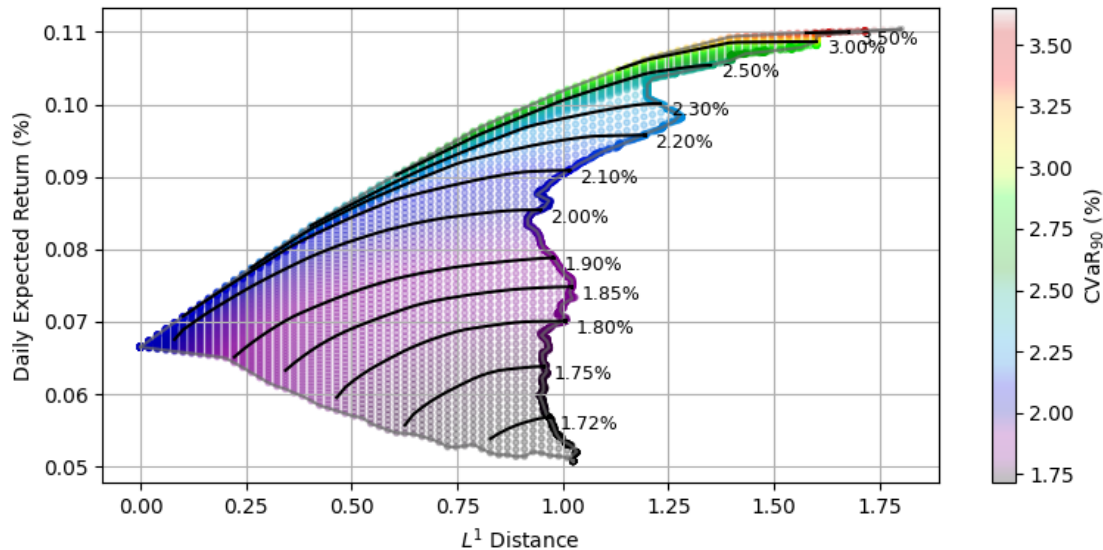
ax.set_xlabel('$L^1$ Distance')
ax.set_ylabel('Daily Expected Return (%)')

```

```

ax.grid(True)
plt.tight_layout()
plt.savefig("cost_contours.pdf", dpi=300, bbox_inches='tight', transparent=True)
plt.show()

```



```

[72]: # portfolios for comparison, stored as 'M5_result_df'
cvar1_level=0.9

records = []

w, cvar1, ret, dis = min_cvar_w_cost(
    returns=stock_returns_subset,
    alpha=cvar1_level,
    max_l1=0.85,
    w0 = start_port,
    target_return=0.06
)
port_rets = stock_returns_subset @ w
cum = (1 + port_rets/100).cumprod()
single_day_max_loss = -port_rets.values.min()
max_dd = max_drawdown(cum.values)*100
records.append({'Target_Return': 0.06, 'Return': ret, 'CVaR': cvar1,
               ↪ 'MaxDailyLoss': single_day_max_loss, 'MaxDrawdown': max_dd, 'Weights': w})

w, cvar1, ret, dis = min_cvar_w_cost(
    returns=stock_returns_subset,
    alpha=cvar1_level,
    max_l1=0.75,

```

```

        w0 = start_port,
        target_return=0.085
    )
port_rets = stock_returns_subset @ w
cum = (1 + port_rets/100).cumprod()
single_day_max_loss = -port_rets.values.min()
max_dd = max_drawdown(cum.values)*100
records.append({'Target_Return': 0.085, 'Return': ret, 'CVaR': cvar1,
               ↪'MaxDailyLoss': single_day_max_loss, 'MaxDrawdown': max_dd, 'Weights': w})

w, cvar1, ret, dis = min_cvar_w_cost(
    returns=stock_returns_subset,
    alpha=cvar1_level,
    max_ll=1.25,
    w0 = start_port,
    target_return=0.105
)
port_rets = stock_returns_subset @ w
cum = (1 + port_rets/100).cumprod()
single_day_max_loss = -port_rets.values.min()
max_dd = max_drawdown(cum.values)*100
records.append({'Target_Return': 0.105, 'Return': ret, 'CVaR': cvar1,
               ↪'MaxDailyLoss': single_day_max_loss, 'MaxDrawdown': max_dd, 'Weights': w})

M5_result_df = pd.DataFrame(records)

```

3 Out-of-Sample

Lastly, we compare performance of chosen portfolios on a new dataset.

First we prepare the data and calculate performance.

```

[73]: # data preparation
START_DATE = '2022-12-13'
END_DATE = '2024-12-10'
YEAR = 252

stock_returns.index = pd.to_datetime(stock_returns.index)

stock_returns_subset_out = stock_returns[(stock_returns.index >= START_DATE) &
                                         (stock_returns.index <= END_DATE)]
outofsample = len(stock_returns_subset_out)
daily_expected_returns_out = stock_returns_subset_out.mean()
assets = len(daily_expected_returns_out)
print(f" For out-of-sample data, we have returns for {outofsample} days.")

```

For out-of-sample data, we have returns for 500 days.

```
[74]: # function that computes all metrics we are interested in
def compute_new_portfolio_metrics(row, returns_df, alpha=0.9):
    w = row['Weights']
    w0=np.ones(10) / 10

    # Portfolio returns and risk: Return & CVaR
    mean_r, cvar = calculate_portfolio_return_and_cvar(returns_df, w, alpha)

    # Compute single-day max loss (most negative daily return)
    daily_returns = returns_df @ w
    single_day_max_loss = -daily_returns.min()
    cum = (1 + daily_returns/100).cumprod()
    max_dd = max_drawdown(cum.values)*100
    dist = l1_distance(w,w0)

    return pd.Series({
        'Return': mean_r,
        'CVaR': cvar,
        'SingleDayMaxLoss': single_day_max_loss,
        'MaxDrawdown': max_dd,
        'Distance': dist
    })
```

```
[75]: # attach the new metrics to MX_result_df
alpha = 0.9

M1_result_df[['Return_out', 'CVaR_out', 'SingleDayMaxLoss_out', 'MaxDrawdown_out', 'Distance']] = M1_result_df.apply(
    compute_new_portfolio_metrics,
    axis=1,
    returns_df=stock_returns_subset_out,
    alpha=alpha
)

M2_result_df[['Return_out', 'CVaR_out', 'SingleDayMaxLoss_out', 'MaxDrawdown_out', 'Distance']] = M2_result_df.apply(
    compute_new_portfolio_metrics,
    axis=1,
    returns_df=stock_returns_subset_out,
    alpha=alpha
)

M3_result_df[['Return_out', 'CVaR_out', 'SingleDayMaxLoss_out', 'MaxDrawdown_out', 'Distance']] = M3_result_df.apply(
    compute_new_portfolio_metrics,
    axis=1,
    returns_df=stock_returns_subset_out,
```

```

        alpha=alpha
    )

M4_result_df[['Return_out', 'CVaR_out', 'SingleDayMaxLoss_out', 'MaxDrawdown_out', 'Distance']] = M4_result_df.apply(
    compute_new_portfolio_metrics,
    axis=1,
    returns_df=stock_returns_subset_out,
    alpha=alpha
)

M5_result_df[['Return_out', 'CVaR_out', 'SingleDayMaxLoss_out', 'MaxDrawdown_out', 'Distance']] = M5_result_df.apply(
    compute_new_portfolio_metrics,
    axis=1,
    returns_df=stock_returns_subset_out,
    alpha=alpha
)

```

Presenting the results using tables and plots.

```

[76]: # plot actual return againsts CVaR_90 of portfolios by M3
targets = M3_result_df['Target_Return'].unique()

groups = M3_result_df.groupby('Target_Return')

plt.rcParams.update({'font.size': 14})

fig, axes = plt.subplots(len(targets), 1, figsize=(6, 4 * len(targets)))
for idx, (ax, (tr, df_group)) in enumerate(zip(axes, groups)):
    x = df_group['CVaR']
    y = df_group['Return_out']
    ax.scatter(x, y, alpha=1, color='C0', label='Expected Return')
    ax.set_xlabel('CVaR$_{90}$ (%)')
    ax.set_ylabel('Out-of-Sample Daily Return (%)', color='C0')
    ax.tick_params(axis='y', labelcolor='C0')
    ax.grid(True)

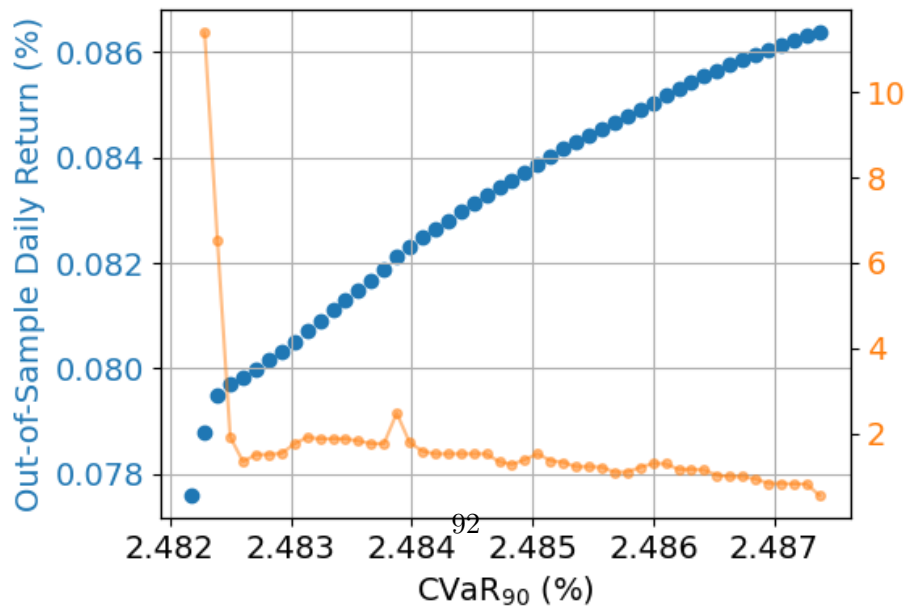
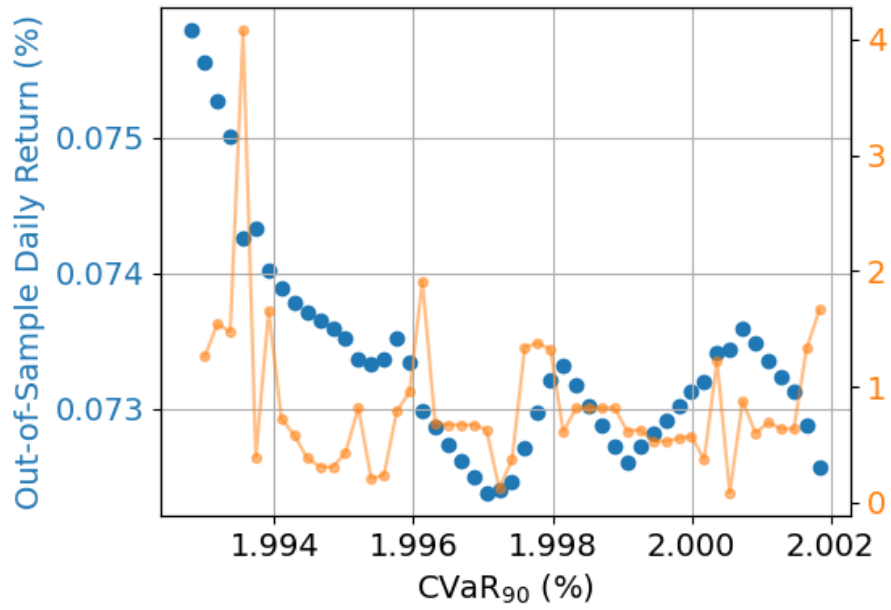
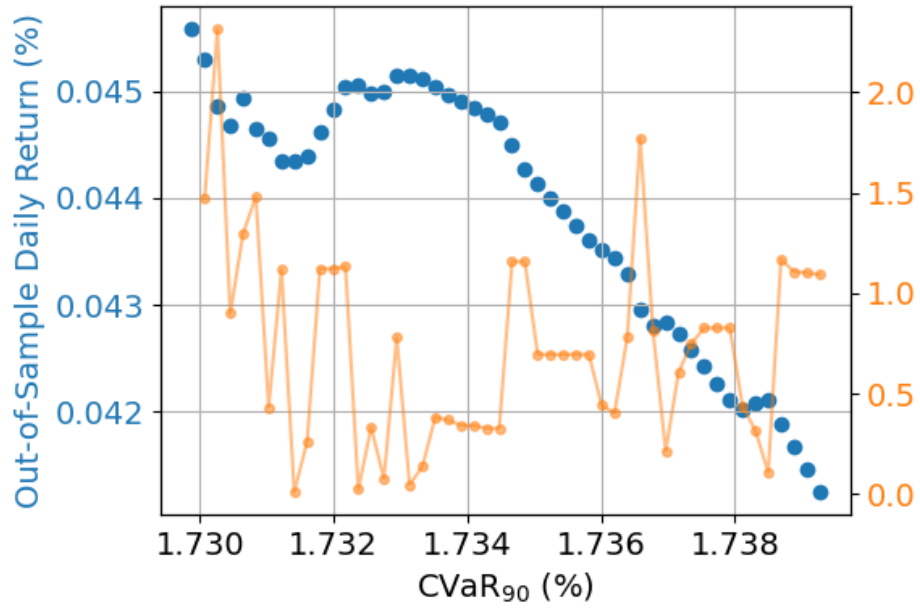
    if idx == 1:
        ax.yaxis.set_major_locator(MaxNLocator(nbins=4, prune='both'))
        ax.yaxis.set_major_formatter(StrMethodFormatter('{x:.3f}'))

    # Compute ratio of changes
    dx = x.diff().to_numpy()[1:]
    dy = y.diff().to_numpy()[1:]
    ratio = abs(dy) / abs(dx)

```

```
# Secondary axis for ratio
ax2 = ax.twinx()
ax2.plot(
    x.to_numpy()[1:], ratio,
    color='C1', marker='o', markersize=4,
    linestyle='-', alpha=0.5
)
ax2.tick_params(axis='y', labelcolor='C1')

plt.tight_layout()
plt.savefig("M3_perf.pdf", dpi=300, bbox_inches='tight', transparent=True)
plt.show()
```



```

[77]: # plot actual return againsts CVaR_90 of portfolios by M4
targets = M4_result_df['Target_Return'].unique()

groups = M4_result_df.groupby('Target_Return')

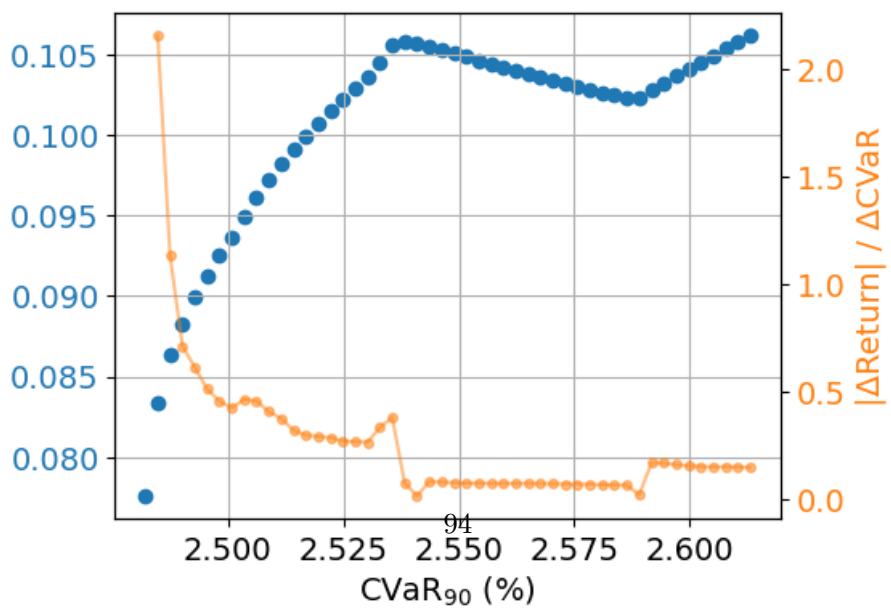
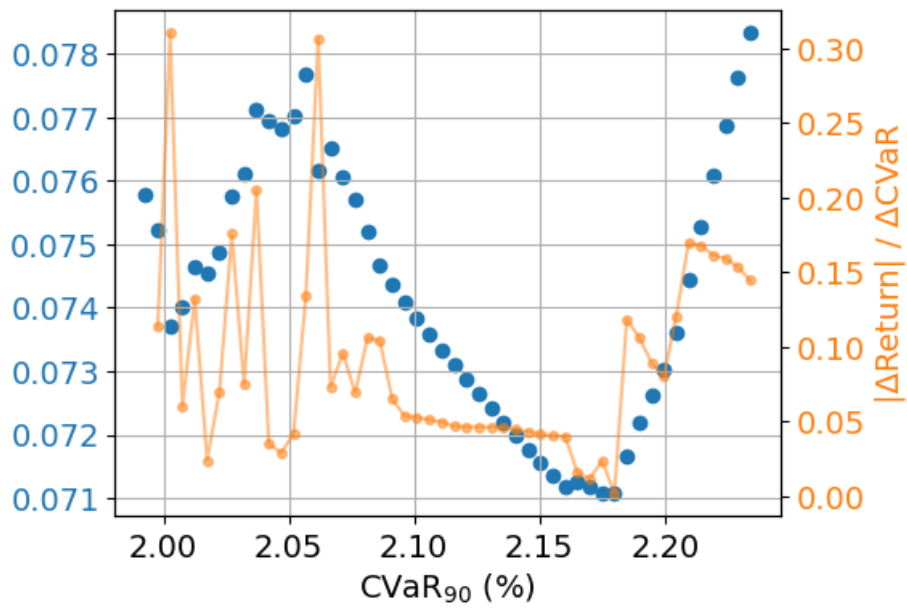
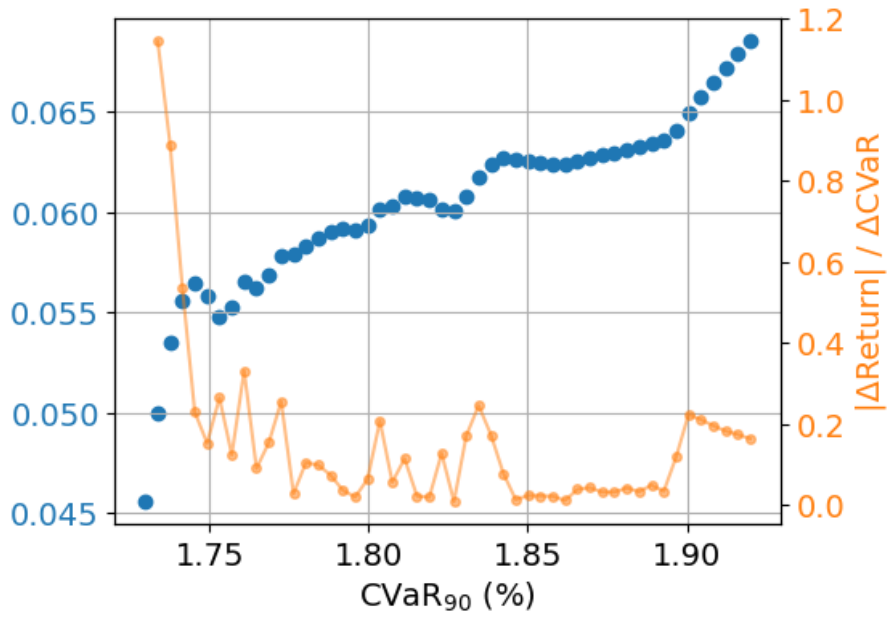
fig, axes = plt.subplots(len(targets), 1, figsize=(6, 4 * len(targets)))
for ax, (tr, df_group) in zip(axes, groups):
    x = df_group['CVaR']
    y = df_group['Return_out']
    ax.scatter(x, y, alpha=1, color='C0')
    ax.set_xlabel('CVaR$_{90}$ (%)')
    ax.tick_params(axis='y', labelcolor='C0')
    ax.grid(True)

    # Compute ratio of changes
    dx = x.diff().to_numpy()[1:]
    dy = y.diff().to_numpy()[1:]
    ratio = abs(dy) / abs(dx)

    # Secondary axis for ratio
    ax2 = ax.twinx()
    ax2.plot(
        x.to_numpy()[1:], ratio,
        color='C1', marker='o', markersize=4,
        linestyle='-', alpha=0.5
    )
    ax2.set_ylabel('|ΔReturn| / ΔCVaR', color='C1')
    ax2.tick_params(axis='y', labelcolor='C1')

plt.tight_layout()
plt.savefig("M4_perf.pdf", dpi=300, bbox_inches='tight', transparent=True)
plt.show()
plt.rcParamsdefaults()

```



```
[78]: # create a table with performance comparison of models from M1, M2, M5 for 0.06
target = 0.06
perf_cols = ["Return", "Return_out", "CVaR", "CVaR_out", "MaxDailyLoss",
↪ "SingleDayMaxLoss_out", "MaxDrawdown",
↪ "MaxDrawdown_out", 'Distance']

# Extract the row for each model
data = {
    "M1": M1_result_df.loc[M1_result_df["Target_Return"] == target, perf_cols].
↪squeeze(),
    "M2": M2_result_df.loc[M2_result_df["Target_Return"] == target, perf_cols].
↪squeeze(),
    "M5": M5_result_df.loc[M5_result_df["Target_Return"] == target, perf_cols].
↪squeeze()
}

perf_df = pd.DataFrame(data, index=perf_cols)

latex_str = perf_df.to_latex(
    float_format="%.3f",
    column_format="lccc",
    bold_rows=True,
    caption=f"Performance metrics at Target Return = {target:.2%}",
    label=f"tab:perf_{int(target*1000)}"
)
print(latex_str)
```

```
\begin{table}
\caption{Performance metrics at Target Return = 6.00%}
\label{tab:perf_60}
\begin{tabular}{lccc}
\toprule
& M1 & M2 & M5 \\
\midrule
\textbf{Return} & 0.060 & 0.060 & 0.060 \\
\textbf{Return_out} & 0.046 & 0.061 & 0.049 \\
\textbf{CVaR} & 1.730 & 1.811 & 1.735 \\
\textbf{CVaR_out} & 1.083 & 1.226 & 1.076 \\
\textbf{MaxDailyLoss} & 7.877 & 6.565 & 7.759 \\
\textbf{SingleDayMaxLoss_out} & 2.275 & 2.392 & 2.333 \\
\textbf{MaxDrawdown} & 27.156 & 22.743 & 27.045 \\
\textbf{MaxDrawdown_out} & 8.921 & 9.601 & 8.826 \\
\textbf{Distance} & 0.949 & 1.033 & 0.850 \\
\bottomrule
\end{tabular}
```

```
\end{table}
```

```
[79]: # create a table with performance comparison of models from M1, M2, M5 for 0.085
target = 0.085
perf_cols = ["Return", "Return_out", "CVaR", "CVaR_out", "MaxDailyLoss",
↪ "SingleDayMaxLoss_out", "MaxDrawdown",
↪ "MaxDrawdown_out", 'Distance']

# Extract the row for each model
data = {
    "M1": M1_result_df.loc[M1_result_df["Target_Return"] == target, perf_cols].
↪squeeze(),
    "M2": M2_result_df.loc[M2_result_df["Target_Return"] == target, perf_cols].
↪squeeze(),
    "M5": M5_result_df.loc[M5_result_df["Target_Return"] == target, perf_cols].
↪squeeze()
}

perf_df2 = pd.DataFrame(data, index=perf_cols)

latex_str = perf_df2.to_latex(
    float_format="%.3f",
    column_format="lccc",
    bold_rows=True,
    caption=f"Performance metrics at Target Return = {target:.2%}",
    label=f"tab:perf_{int(target*1000)}"
)
print(latex_str)
```

```
\begin{table}
\caption{Performance metrics at Target Return = 8.50%}
\label{tab:perf_85}
\begin{tabular}{lccc}
\toprule
& M1 & M2 & M5 \\
\midrule
\textbf{Return} & 0.085 & 0.085 & 0.085 \\
\textbf{Return_out} & 0.076 & 0.075 & 0.077 \\
\textbf{CVaR} & 1.993 & 2.022 & 2.007 \\
\textbf{CVaR_out} & 1.372 & 1.310 & 1.322 \\
\textbf{MaxDailyLoss} & 10.364 & 9.822 & 10.507 \\
\textbf{SingleDayMaxLoss_out} & 2.902 & 3.013 & 3.020 \\
\textbf{MaxDrawdown} & 25.567 & 24.333 & 26.338 \\
\textbf{MaxDrawdown_out} & 8.452 & 8.524 & 7.664 \\
\textbf{Distance} & 0.926 & 1.000 & 0.750 \\
\bottomrule
\end{tabular}
```

```
\end{table}
```

```
[80]: # create a table with performance comparison of models from M1, M2, M5 for 0.105
target = 0.105
perf_cols = ["Return", "Return_out", "CVaR", "CVaR_out", "MaxDailyLoss",
            ↪ "SingleDayMaxLoss_out", "MaxDrawdown",
            ↪ "MaxDrawdown_out", 'Distance']

# Extract each model's row
data = {
    "M1": M1_result_df.loc[M1_result_df["Target_Return"] == target, perf_cols].
    ↪squeeze(),
    "M2": M2_result_df.loc[M2_result_df["Target_Return"] == target, perf_cols].
    ↪squeeze(),
    "M5": M5_result_df.loc[M5_result_df["Target_Return"] == target, perf_cols].
    ↪squeeze()
}
perf_df3 = pd.DataFrame(data, index=perf_cols)

latex_str = perf_df3.to_latex(
    float_format="%.3f",
    column_format="lccc",
    bold_rows=True,
    caption=f"Performance metrics at Target Return = {target:.2%}",
    label=f"tab:perf_{int(target*1000)}"
)
print(latex_str)
```

```
\begin{table}
\caption{Performance metrics at Target Return = 10.50%}
\label{tab:perf_105}
\begin{tabular}{lccc}
\toprule
& M1 & M2 & M5 \\
\midrule
\textbf{Return} & 0.105 & 0.105 & 0.105 \\
\textbf{Return_out} & 0.078 & 0.099 & 0.105 \\
\textbf{CVaR} & 2.482 & 2.514 & 2.529 \\
\textbf{CVaR_out} & 1.814 & 1.713 & 1.781 \\
\textbf{MaxDailyLoss} & 13.892 & 12.450 & 11.858 \\
\textbf{SingleDayMaxLoss_out} & 4.630 & 3.787 & 3.700 \\
\textbf{MaxDrawdown} & 29.909 & 27.112 & 25.661 \\
\textbf{MaxDrawdown_out} & 9.990 & 10.120 & 9.728 \\
\textbf{Distance} & 1.318 & 1.272 & 1.250 \\
\bottomrule
\end{tabular}
\end{table}
```

