foam

January 5, 2025

1 Mean-Variance Model

1.1 Exercise 2.

- Task 0: Cleaning Data and Importing Libraries:

[3]:
```
pip install gurobipy
```

```
Collecting gurobipy
  Downloading gurobipy-12.0.0-cp312-cp312-
manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (15 kB)
Downloading
gurobipy-12.0.0-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (14.2
MB)
                         14.2/14.2 MB
69.2 MB/s eta 0:00:00:00:01
Installing collected packages: gurobipy
Successfully installed gurobipy-12.0.0
Note: you may need to restart the kernel to use updated packages.
```

[7]:
```
pip install openpyxl
```

```
Collecting openpyxl
  Downloading openpyxl-3.1.5-py2.py3-none-any.whl.metadata (2.5 kB)
Collecting et-xmlfile (from openpyxl)
  Downloading et_xmlfile-2.0.0-py3-none-any.whl.metadata (2.7 kB)
Downloading openpyxl-3.1.5-py2.py3-none-any.whl (250 kB)
Downloading et_xmlfile-2.0.0-py3-none-any.whl (18 kB)
Installing collected packages: et-xmlfile, openpyxl
Successfully installed et-xmlfile-2.0.0 openpyxl-3.1.5
Note: you may need to restart the kernel to use updated packages.
```

[5]:
```
#import required libraries:
import gurobipy as gb
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

I used gurobipy library to optimize the portfolio, the other libraries are for data manipulations and plotting the graphs. We first need to make sure we correctly loaded the data.

[8]:
```
df=pd.read_excel("/workspaces/FOAM/data/data.xlsx", sheet_name="Returns S&P Mib␣
↪30 ", index_col=0, parse_dates=True)
df.head(2)
```

[8]:
```
                  A1        A2        A3        A4        A5        A6  \
Dates
2004-01-01  0.056941  0.055085 -0.005061 -0.001376  0.038502  0.004092
2004-02-01 -0.025385 -0.097992 -0.083576 -0.046853 -0.041082  0.020377

                  A7        A8        A9       A10       A11       A12  \
Dates
2004-01-01 -0.003996 -0.023231  0.036722 -0.018517  0.023003  0.008654
2004-02-01 -0.064916 -0.025405 -0.035421 -0.140655 -0.066209 -0.094376

                 A13       A14       A15       A16       A17       A18  \
Dates
2004-01-01 -0.025495 -0.028095  0.134228 -0.035847  0.039152 -0.031417
2004-02-01 -0.037007 -0.009937  0.064431 -0.106229 -0.009411 -0.077847

                 A19       A20
Dates
2004-01-01  0.027356 -0.027555
2004-02-01  0.004931 -0.083825
```

We can drop the index name, and change the date format appropriately (back to the given format)

[9]:
```
df.index.name=None
df.index = df.index.strftime('%b-%Y')
df.head(2)
```

[9]:
```
               A1        A2        A3        A4        A5        A6  \
Jan-2004  0.056941  0.055085 -0.005061 -0.001376  0.038502  0.004092
Feb-2004 -0.025385 -0.097992 -0.083576 -0.046853 -0.041082  0.020377

               A7        A8        A9       A10       A11       A12  \
Jan-2004 -0.003996 -0.023231  0.036722 -0.018517  0.023003  0.008654
Feb-2004 -0.064916 -0.025405 -0.035421 -0.140655 -0.066209 -0.094376

              A13       A14       A15       A16       A17       A18  \
Jan-2004 -0.025495 -0.028095  0.134228 -0.035847  0.039152 -0.031417
Feb-2004 -0.037007 -0.009937  0.064431 -0.106229 -0.009411 -0.077847

              A19       A20
Jan-2004  0.027356 -0.027555
Feb-2004  0.004931 -0.083825
```

Since the stocks are given in the range from A1 to A20, I prepared dictionary to replace them with their respective names. So that we can plot the efficient frontier including the real asset names.

[10]:
```
stocks = [
    "AL", "AGL", "AUTO", "NTV", "BFI", "BIN", "BPM", "BPVN", "BNL", "BPU",␣
 ↪"BUL", "CAP",
    "EDN", "ENEL", "ENI", "FWB", "F", "FNC", "G", "ES"
] # we can later use this list to refer to the columns of the dataset.

stock_dict = {f"A{i+1}": stock for i, stock in enumerate(stocks)}

# now we can replace them
df = df.rename(columns=stock_dict)
df.head(2)
```

[10]:
```
                AL       AGL      AUTO       NTV       BFI       BIN  \
Jan-2004  0.056941  0.055085 -0.005061 -0.001376  0.038502  0.004092
Feb-2004 -0.025385 -0.097992 -0.083576 -0.046853 -0.041082  0.020377

               BPM      BPVN       BNL       BPU       BUL       CAP  \
Jan-2004 -0.003996 -0.023231  0.036722 -0.018517  0.023003  0.008654
Feb-2004 -0.064916 -0.025405 -0.035421 -0.140655 -0.066209 -0.094376

               EDN      ENEL       ENI       FWB         F       FNC  \
Jan-2004 -0.025495 -0.028095  0.134228 -0.035847  0.039152 -0.031417
Feb-2004 -0.037007 -0.009937  0.064431 -0.106229 -0.009411 -0.077847

                 G        ES
Jan-2004  0.027356 -0.027555
Feb-2004  0.004931 -0.083825
```

1.2 Exercise 2.1

- Task 1: Compute the expected returns of all assets in the market and the matrix of variances and covariances.
- Task 2: Formulate and solve the Markowitz model for finding the maximum possible expected return value for an efficient portfolio (ER_max)
- Task 3: Formulate and solve the Markowitz model to find the minimum possible expected return value for an efficient portfolio (ER_min).
- Task 4: Fix 5 different values for the portfolio expected return in the range (ER_min, ER_max), denoting them by , , , , .

1.2.1 Task 1: Compute the expected returns of all assets in the market and the matrix of variances and covariances

Expected returns are found as follows, since the data frame is already about the returns we can just take the `.mean()` (mean) for an asset for the whole period.

[11]:
```
expected_returns = df[stocks].mean()
expected_returns.head(2)
```

[11]:
```
AL     0.001935
AGL    0.028860
dtype: float64
```

Compute variance and covariance matrix with `.cov()` function, where main diagnal is variance and the rest are covariance values.

[12]:
```
covariance_matrix = df[stocks].cov()
covariance_matrix.head(2)
```

[12]:
```
           AL       AGL      AUTO       NTV       BFI       BIN       BPM  \
AL   0.005113  0.000429  0.001942  0.000599  0.000211  0.000246  0.000022
AGL  0.000429  0.006196  0.002060  0.001682  0.001280  0.001742  0.002319

         BPVN       BNL       BPU       BUL       CAP       EDN      ENEL  \
AL  -0.000506  0.000008  0.001111  0.000096  0.001030  0.001152  0.000063
AGL  0.001718  0.002281  0.002289  0.001349  0.001539  0.001463  0.001753

          ENI       FWB         F       FNC         G        ES
AL   0.000536  0.001054  0.000214  0.003113  0.001102  0.000578
AGL  0.001833  0.001800  0.001016  0.002068  0.001396  0.000841
```

---

2 Model Implementation

2.0.1 Task 2: Formulate and solve the Markowitz model for finding the maximum possible expected return value for an efficient portfolio (ER_{max})

Here, since we want to find ER_{max}, we need to set the objective function to `MAXIMIZE` the portfolio return.

[13]:
```
# Create model
model_one = gb.Model()

# Add variables for portfolio weights
x = pd.Series(model_one.addVars(stocks, lb=0, name='X'), index=stocks)
```

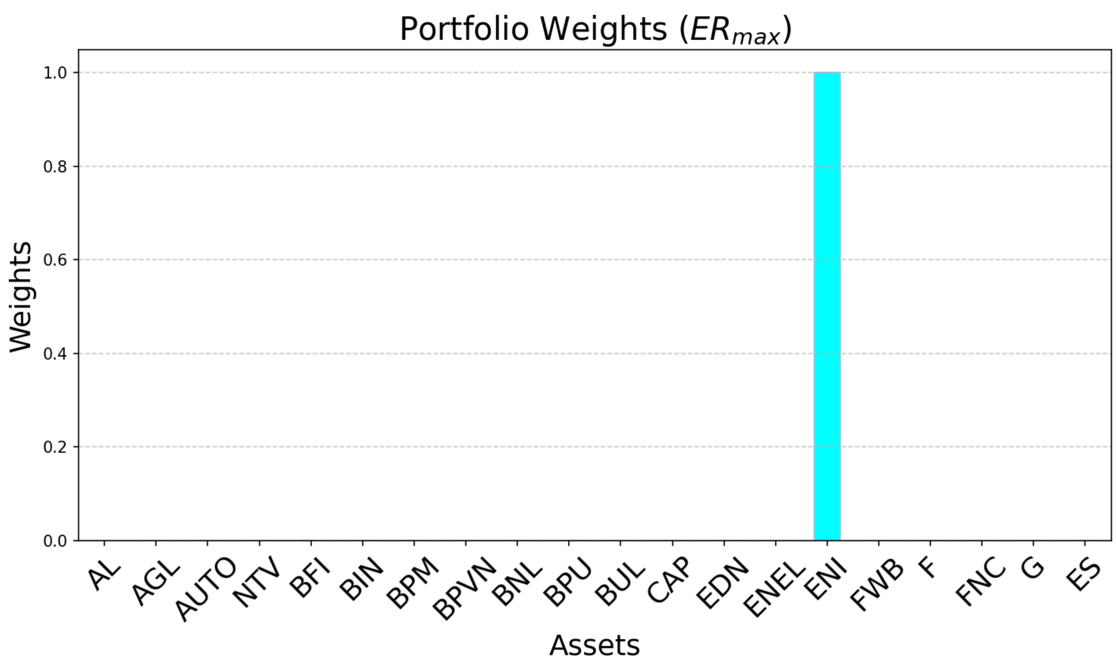
```python
# Compute portfolio variance (to use as constraint) and return (to use in␣
↪objective funtion)
portfolio_variance = covariance_matrix.dot(x).dot(x)
portfolio_return = expected_returns.dot(x)

# Add objective function: Maximize portfolio return
model_one.setObjective(portfolio_return, sense=gb.GRB.MAXIMIZE)

# Add budget constraint: Sum of weights equals 1
model_one.addConstr(x.sum() == 1, name="budget")

# Add portfolio variance constraint: Variance less than or equal to␣
↪sigma_squared
# Here I relaxed the constraint because if we fix the sigma, the model could␣
↪not find
# the optimal solution and the output was infeasable or unbound.
sigma_squared = 2
model_one.addConstr(portfolio_variance <= sigma_squared,␣
↪name="variance_constraint")

# Optimize model
model_one.optimize()

# Display results
if model_one.status == gb.GRB.OPTIMAL:
    model_one_weights = pd.Series({stock: x[stock].X for stock in stocks})
    print("Optimal Portfolio Weights:")
    print(model_one_weights)
    print(f"ER_max: {portfolio_return.getValue()}")
    print(f"Portfolio Variance: {portfolio_variance.getValue()}")
else:
    print("No feasible solution found.")
```

```
Restricted license - for non-production use only - expires 2026-11-23
Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (linux64 - "Ubuntu 20.04.6
LTS")

CPU model: Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 1 rows, 20 columns and 20 nonzeros
Model fingerprint: 0x7ef9aa5b
Model has 1 quadratic constraint
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  QMatrix range    [2e-05, 6e-03]
```

```
  Objective range  [3e-04, 3e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
  QRHS range       [2e+00, 2e+00]
Presolve time: 0.02s
Presolved: 1 rows, 20 columns, 20 nonzeros
Ordering time: 0.00s

Barrier statistics:
 AA' NZ     : 0.000e+00
 Factor NZ  : 1.000e+00
 Factor Ops : 1.000e+00 (less than 1 second per iteration)
 Threads    : 1

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   4.47337501e-02  1.07226095e-02  3.17e+00 3.64e-01  1.56e-01     0s
   1   1.50672255e-02  4.00025945e-02  1.33e-15 4.94e-17  1.99e-02     0s
   2   3.01017861e-02  3.41526871e-02  2.22e-16 1.53e-16  3.24e-03     0s
   3   3.34611277e-02  3.35365587e-02  0.00e+00 5.55e-17  6.03e-05     0s
   4   3.34857476e-02  3.34858230e-02  0.00e+00 1.11e-16  6.03e-08     0s
   5   3.34857719e-02  3.34857719e-02  2.22e-16 1.53e-16  6.03e-14     0s

Barrier solved model in 5 iterations and 0.04 seconds (0.00 work units)
Optimal objective 3.34857719e-02

Optimal Portfolio Weights:
AL      4.194367e-15
AGL     3.125172e-12
AUTO    2.455500e-14
NTV     2.533210e-14
BFI     2.422741e-15
BIN     2.735301e-13
BPM     4.326931e-14
BPVN    3.243653e-15
BNL     8.972285e-14
BPU     5.064772e-15
BUL     3.696030e-15
CAP     8.393205e-14
EDN     3.997134e-15
ENEL    1.733407e-14
ENI     1.000000e+00
FWB     3.910363e-14
F       2.431586e-14
FNC     1.555221e-15
G       3.016431e-15
ES      2.453722e-14
dtype: float64
```

```
ER_max: 0.03348577185008575
Portfolio Variance: 0.004442189918950766
```

[14]:
```python
plt.figure(figsize=(10, 6), dpi=150)
model_one_weights.plot(kind="bar", color="cyan", edgecolor="skyblue")
plt.title(r"Portfolio Weights ($ER_{max}$)", fontsize=20)
plt.xlabel("Assets", fontsize=18)
plt.ylabel("Weights", fontsize=18)
plt.xticks(rotation=45, fontsize=18)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.tight_layout()
# plt.savefig("/Users/javlon/Documents/GitHub/RedPill/FOAM/Analysis/
↪Portfolio_Weights_for_ER_max", dpi=300)
plt.show()
```



2.0.2 Task 3: Formulate and solve the Markowitz model to find the minimum possible expected return value for an efficient portfolio (ER_min).

But in this task, we are asked to find ER_{min}, so we need to set the objective function to `MINIMIZE` the portfolio return.

[15]:
```python
model_two = gb.Model()

x = pd.Series(model_two.addVars(stocks, lb=0, name='X'), index=stocks)

portfolio_variance = covariance_matrix.dot(x).dot(x)
```

```python
portfolio_return = expected_returns.dot(x)

# everything is the same as above except this part, where maximize is replaced␣
↪with minimize
model_two.setObjective(portfolio_return, sense=gb.GRB.MINIMIZE)

model_two.addConstr(x.sum() == 1, name="budget")

sigma_squared = 2
model_two.addConstr(portfolio_variance <= sigma_squared,␣
↪name="variance_constraint")

model_two.optimize()

if model_two.status == gb.GRB.OPTIMAL:
    model_two_weights = pd.Series({stock: x[stock].X for stock in stocks})
    print("Optimal Portfolio Weights:")
    print(model_two_weights)
    print(f"ER_min: {portfolio_return.getValue()}")
    print(f"Portfolio Variance: {portfolio_variance.getValue()}")
else:
    print("No feasible solution found.")
```

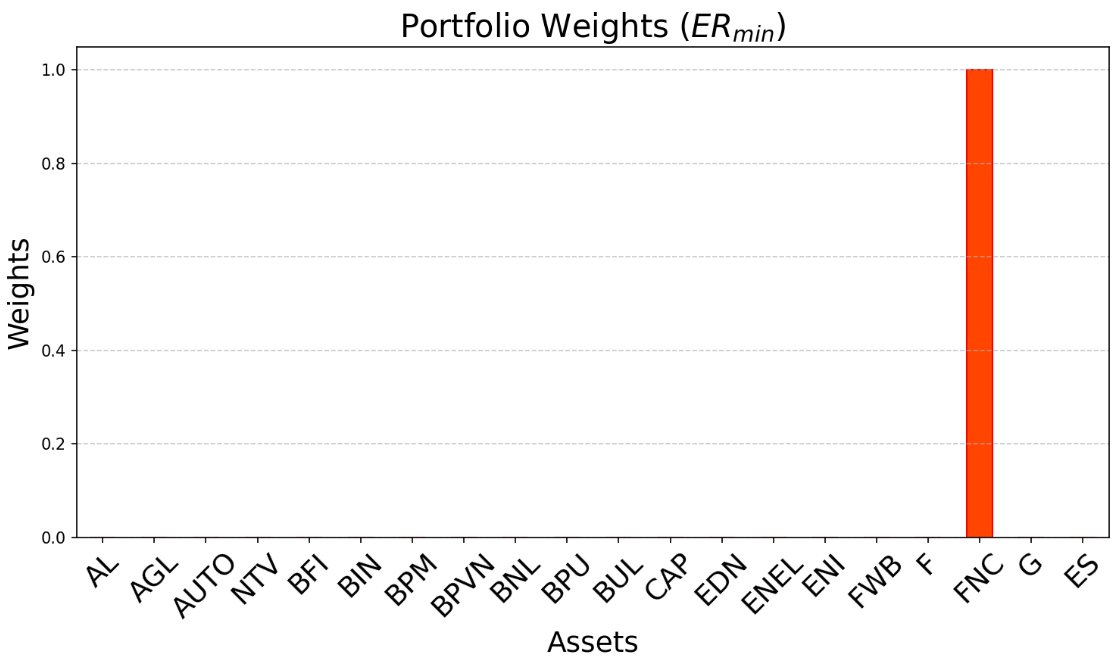
```
Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (linux64 - "Ubuntu 20.04.6
LTS")

CPU model: Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 1 rows, 20 columns and 20 nonzeros
Model fingerprint: 0x4031cfc0
Model has 1 quadratic constraint
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  QMatrix range    [2e-05, 6e-03]
  Objective range  [3e-04, 3e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
  QRHS range       [2e+00, 2e+00]
Presolve time: 0.05s
Presolved: 1 rows, 20 columns, 20 nonzeros
Ordering time: 0.00s

Barrier statistics:
 AA' NZ     : 0.000e+00
 Factor NZ  : 1.000e+00
```

```
 Factor Ops : 1.000e+00 (less than 1 second per iteration)
 Threads    : 1

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   4.47337501e-02  0.00000000e+00  3.17e+00 1.51e-01  1.20e-01     0s
   1   6.11364084e-03 -1.30002487e-02  6.66e-16 1.11e-16  1.53e-02     0s
   2  -4.11407765e-03 -1.09577111e-02  2.22e-16 2.78e-16  5.47e-03     0s
   3  -9.23160909e-03 -9.62378607e-03  2.22e-16 1.11e-16  3.14e-04     0s
   4  -9.40868960e-03 -9.40908902e-03  0.00e+00 1.11e-16  3.20e-07     0s
   5  -9.40886788e-03 -9.40886828e-03  0.00e+00 1.11e-16  3.20e-10     0s

Barrier solved model in 5 iterations and 0.10 seconds (0.00 work units)
Optimal objective -9.40886788e-03

Optimal Portfolio Weights:
AL      1.193706e-09
AGL     3.767788e-11
AUTO    1.490483e-10
NTV     1.438027e-10
BFI     8.036580e-09
BIN     4.489211e-11
BPM     8.854353e-11
BPVN    2.255074e-09
BNL     5.992037e-11
BPU     8.345060e-10
BUL     1.589626e-09
CAP     6.154941e-11
EDN     1.324457e-09
ENEL    2.075137e-10
ENI     3.710248e-11
FWB     9.548225e-11
F       1.503467e-10
FNC     1.000000e+00
G       2.826122e-09
ES      1.491638e-10
dtype: float64
ER_min: -0.009408867882700437
Portfolio Variance: 0.005363939800538677
```

[16]:

```python
plt.figure(figsize=(10, 6), dpi=150)
model_two_weights.plot(kind="bar", color="orangered", edgecolor="red")
plt.title(r"Portfolio Weights ($ER_{min}$)", fontsize=20)
plt.xlabel("Assets", fontsize=18)
plt.ylabel("Weights", fontsize=18)
plt.xticks(rotation=45, fontsize=18)
plt.grid(axis="y", linestyle="--", alpha=0.7)
```

```
plt.tight_layout()
# plt.savefig("/Users/javlon/Documents/GitHub/RedPill/FOAM/Analysis/
↪Portfolio_Weights_for_ER_min", dpi=300)
plt.show()
```



2.0.3 Task 4: Fix 5 different values for the portfolio expected return in the range (ER_min, ER_max), denoting them by , , , , .

We have already found values for ER_{min} ( ) and ER_{max} ( ), we now need to another 3 values for µ's so that there will be 5 equally spaced portfolio returns.

[17]:
```
ER_min = -0.009408867882700435
ER_max = 0.03348577185008574

mu_values = np.linspace(ER_min, ER_max, 5)
mu1, mu2, mu3, mu4, mu5 = mu_values

print(f"mu1: {mu1:.8f}")
print(f"mu2: {mu2:.8f}")
print(f"mu3: {mu3:.8f}")
print(f"mu4: {mu4:.8f}")
print(f"mu5: {mu5:.8f}")
```

```
mu1: -0.00940887
mu2: 0.00131479
mu3: 0.01203845
```

```
mu4: 0.02276211
mu5: 0.03348577
```

2.1 Exercise 2.2

- Solve model (MwV) formulated in 1.1 for each possible value $\mu_1, \mu_2, \mu_3, \mu_4, \mu_5$. Denote by $P_{\mu 1}, P_{\mu 2}, P_{\mu 3}, P_{\mu 4}, P_{\mu 5}$ the optimal portfolios so found.

So above, when we find the $ER_{min} and ER_{max}$, we input the portfolio variance as constraint and the portfolio return as objective function, here we switch the places.

[18]:
```
# Define the list of mu values
mu_values = [mu1, mu2, mu3, mu4, mu5]

# Create a dictionary to store results
m3_results = {}

# List to store data for plotting the efficient frontier
m3_efficient_frontier = []

for i, mu in enumerate(mu_values, 1):
    # Create a new model
    m3 = gb.Model()

    # Add variables for portfolio weights
    x = pd.Series(m3.addVars(stocks, lb=0, name='X'), index=stocks)

    # Compute portfolio variance and return
    portfolio_variance = covariance_matrix.dot(x).dot(x)
    portfolio_return = expected_returns.dot(x)

    # Add objective function: Minimize portfolio variance
    m3.setObjective(portfolio_variance, sense=gb.GRB.MINIMIZE)

    # Add budget constraint: Sum of weights equals 1
    m3.addConstr(x.sum() == 1, name="budget")

    # Add portfolio return constraint
    m3_trc = m3.addConstr(portfolio_return == mu, name="return_constraint")

    # Optimize model
    m3.optimize()

    # Store results if feasible solution found
    if m3.status == gb.GRB.OPTIMAL:
        m3_weights = pd.Series({stock: x[stock].X for stock in stocks})
        m3_portfolio_variance_value = portfolio_variance.getValue()
```

```python
        # Store results in a dictionary
        m3_results[f"P_mu_{i}"] = {
            "mu_value": mu,
            "m3_weights": m3_weights,
            "m3_portfolio_variance": m3_portfolio_variance_value,
        }

        # Add data to the efficient frontier list
        m3_efficient_frontier.append({
            "mu": mu,
            "m3_variance": m3_portfolio_variance_value,
            "m3_weights": m3_weights,
        })
    else:
        m3_results[f"P_mu_{i}"] = "No feasible solution found"

# Display results
for key, result in m3_results.items():
    if isinstance(result, dict):
        print(f"\nResults for {key}:")
        print(f"mu value: {result['mu_value']:.8f}")
        print(f"{key}:")
        print(result['m3_weights'])
        print(f"Portfolio Variance: {result['m3_portfolio_variance']:.8f}")
    else:
        print(f"\nResults for {key}: {result}")
```

```
Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (linux64 - "Ubuntu 20.04.6
LTS")

CPU model: Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 2 rows, 20 columns and 40 nonzeros
Model fingerprint: 0xc5acecf4
Model has 210 quadratic objective terms
Coefficient statistics:
  Matrix range     [3e-04, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [3e-05, 1e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [9e-03, 1e+00]
Presolve time: 0.01s
Presolved: 2 rows, 20 columns, 40 nonzeros
Presolved model has 210 quadratic objective terms
Ordering time: 0.00s
```

```
Barrier statistics:
 Free vars  : 19
 AA' NZ     : 2.100e+02
 Factor NZ  : 2.310e+02
 Factor Ops : 3.311e+03 (less than 1 second per iteration)
 Threads    : 1

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   9.18938432e+03 -9.18938432e+03  1.85e+04 5.25e-02  1.00e+06     0s
   1   7.12976742e+02 -7.30604648e+02  7.59e+02 2.15e-03  4.31e+04     0s
   2   2.64422505e-02 -2.08790579e+01  2.47e+00 7.00e-06  1.74e+02     0s
   3   1.92157794e-03 -4.18023795e-02  5.77e-01 1.64e-06  3.71e+01     0s
   4   5.31624913e-03 -1.42991439e-02  1.10e-02 3.13e-08  7.60e-01     0s
   5   5.36389206e-03  5.32428114e-03  1.11e-05 3.14e-11  7.94e-04     0s
   6   5.36393984e-03  5.36370003e-03  6.81e-09 1.93e-14  8.33e-07     0s
   7   5.36393975e-03  5.36383332e-03  1.29e-08 9.09e-13  9.13e-08     0s
   8   5.36393995e-03  5.36383345e-03  1.43e-09 2.27e-13  9.14e-11     0s

Barrier solved model in 8 iterations and 0.03 seconds (0.00 work units)
Optimal objective 5.36393995e-03

Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (linux64 - "Ubuntu 20.04.6
LTS")

CPU model: Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 2 rows, 20 columns and 40 nonzeros
Model fingerprint: 0x5ce81d1c
Model has 210 quadratic objective terms
Coefficient statistics:
  Matrix range     [3e-04, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [3e-05, 1e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e-03, 1e+00]
Presolve time: 0.01s
Presolved: 2 rows, 20 columns, 40 nonzeros
Presolved model has 210 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 19
 AA' NZ     : 2.100e+02
 Factor NZ  : 2.310e+02
```

```
 Factor Ops : 3.311e+03 (less than 1 second per iteration)
 Threads    : 1

                 Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   9.18970999e+03 -9.18970999e+03  1.85e+04 2.24e-04  1.00e+06     0s
   1   7.13054752e+02 -7.29865750e+02  7.59e+02 9.19e-06  4.31e+04     0s
   2   2.92867656e-02 -1.93446881e+01  2.96e+00 3.59e-08  2.00e+02     0s
   3   1.30077221e-03 -1.02856764e+01  8.68e-02 1.05e-09  2.04e+01     0s
   4   1.29256907e-03 -9.32113703e-01  8.68e-08 1.05e-15  1.49e+00     0s
   5   1.29011410e-03 -1.55943322e-03  1.78e-10 1.39e-17  4.56e-03     0s
   6   1.03318777e-03  6.42844087e-04  9.86e-13 1.39e-17  6.25e-04     0s
   7   8.87521757e-04  7.56068517e-04  1.28e-15 5.55e-17  2.10e-04     0s
   8   8.50123464e-04  8.45344635e-04  3.33e-16 3.47e-17  7.65e-06     0s
   9   8.47251817e-04  8.47193642e-04  3.77e-15 4.16e-17  9.31e-08     0s
  10   8.47234364e-04  8.47232642e-04  3.89e-14 7.40e-17  2.76e-09     0s

Barrier solved model in 10 iterations and 0.03 seconds (0.00 work units)
Optimal objective 8.47234364e-04

Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (linux64 - "Ubuntu 20.04.6
LTS")

CPU model: Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 2 rows, 20 columns and 40 nonzeros
Model fingerprint: 0x713ac6af
Model has 210 quadratic objective terms
Coefficient statistics:
  Matrix range     [3e-04, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [3e-05, 1e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e-02, 1e+00]
Presolve time: 0.01s
Presolved: 2 rows, 20 columns, 40 nonzeros
Presolved model has 210 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 19
 AA' NZ     : 2.100e+02
 Factor NZ  : 2.310e+02
 Factor Ops : 3.311e+03 (less than 1 second per iteration)
 Threads    : 1
```

```
                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   9.19003592e+03 -9.19003592e+03  1.85e+04 5.30e-02  1.00e+06     0s
   1   7.13133174e+02 -7.29124464e+02  7.60e+02 2.17e-03  4.31e+04     0s
   2   3.28065817e-02 -1.77583466e+01  3.46e+00 9.90e-06  2.26e+02     0s
   3   1.35522067e-03 -1.35688641e+01  3.46e-06 9.90e-12  2.17e+01     0s
   4   1.35485501e-03 -1.46040080e-02  6.09e-10 1.73e-15  2.55e-02     0s
   5   1.13833530e-03 -6.80095783e-04  4.48e-11 1.25e-16  2.91e-03     0s
   6   7.27795191e-04 -1.19524350e-03  6.66e-16 2.78e-17  3.08e-03     0s
   7   6.04918366e-04  3.68245410e-04  6.94e-17 2.78e-17  3.79e-04     0s
   8   5.51967677e-04  4.80800612e-04  1.33e-15 6.94e-18  1.14e-04     0s
   9   5.32949895e-04  5.26970661e-04  1.39e-16 3.94e-17  9.57e-06     0s
  10   5.30430010e-04  5.30324878e-04  8.88e-16 1.39e-17  1.68e-07     0s
  11   5.30369665e-04  5.30362698e-04  7.32e-14 3.50e-17  1.11e-08     0s
  12   5.30367293e-04  5.30366287e-04  6.31e-13 1.39e-17  1.61e-09     0s

Barrier solved model in 12 iterations and 0.03 seconds (0.00 work units)
Optimal objective 5.30367293e-04

Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (linux64 - "Ubuntu 20.04.6
LTS")

CPU model: Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 2 rows, 20 columns and 40 nonzeros
Model fingerprint: 0x28955c11
Model has 210 quadratic objective terms
Coefficient statistics:
  Matrix range     [3e-04, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [3e-05, 1e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e-02, 1e+00]
Presolve time: 0.01s
Presolved: 2 rows, 20 columns, 40 nonzeros
Presolved model has 210 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 19
 AA' NZ     : 2.100e+02
 Factor NZ  : 2.310e+02
 Factor Ops : 3.311e+03 (less than 1 second per iteration)
 Threads    : 1

                  Objective                Residual
```

```
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   9.19036211e+03 -9.19036211e+03  1.85e+04 1.06e-01  1.00e+06     0s
   1   7.13212007e+02 -7.28380791e+02  7.60e+02 4.34e-03  4.31e+04     0s
   2   3.65631686e-02 -1.61234224e+01  3.93e+00 2.24e-05  2.50e+02     0s
   3   1.69366839e-03 -1.01536087e+01  8.84e-03 5.05e-08  1.67e+01     0s
   4   1.68451532e-03 -1.21209246e-02  9.97e-07 5.70e-12  2.21e-02     0s
   5   1.48658246e-03  2.46004869e-04  4.12e-08 2.35e-13  1.98e-03     0s
   6   1.10300358e-03  5.99606645e-04  4.01e-14 1.39e-17  8.05e-04     0s
   7   9.98283075e-04  9.24545764e-04  8.92e-16 6.94e-18  1.18e-04     0s
   8   9.65556597e-04  9.53763301e-04  1.05e-15 5.55e-17  1.89e-05     0s
   9   9.58912915e-04  9.58729017e-04  1.78e-15 6.94e-17  2.94e-07     0s
  10   9.58807209e-04  9.58806986e-04  2.53e-15 3.99e-17  3.57e-10     0s

Barrier solved model in 10 iterations and 0.04 seconds (0.00 work units)
Optimal objective 9.58807209e-04

Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (linux64 - "Ubuntu 20.04.6
LTS")

CPU model: Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 2 rows, 20 columns and 40 nonzeros
Model fingerprint: 0x1a7d471c
Model has 210 quadratic objective terms
Coefficient statistics:
  Matrix range     [3e-04, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [3e-05, 1e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [3e-02, 1e+00]
Presolve time: 0.03s
Presolved: 2 rows, 20 columns, 40 nonzeros
Presolved model has 210 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 19
 AA' NZ     : 2.100e+02
 Factor NZ  : 2.310e+02
 Factor Ops : 3.311e+03 (less than 1 second per iteration)
 Threads    : 1

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   9.19068856e+03 -9.19068856e+03  1.85e+04 1.58e-01  1.00e+06     0s
```

```
   1   7.13216227e+02 -7.27559915e+02  7.60e+02 6.51e-03  4.32e+04     0s
   2   4.11704947e-02 -1.44404647e+01  4.41e+00 3.78e-05  2.75e+02     0s
   3   3.03481871e-03 -2.19850291e+00  2.10e-01 1.80e-06  1.53e+01     0s
   4   4.42316932e-03 -3.38667944e-02  1.53e-03 1.31e-08  1.60e-01     0s
   5   4.44217075e-03  4.36525025e-03  1.54e-06 1.32e-11  2.22e-04     0s
   6   4.44218990e-03  4.44203607e-03  1.54e-09 1.32e-14  3.45e-07     0s
   7   4.44218992e-03  4.44218974e-03  1.76e-12 6.82e-13  3.46e-10     0s

Barrier solved model in 7 iterations and 0.06 seconds (0.00 work units)
Optimal objective 4.44218992e-03


Results for P_mu_1:
mu value: -0.00940887
P_mu_1:
AL      2.328000e-13
AGL     1.606037e-13
AUTO    2.508947e-13
NTV     2.488702e-13
BFI     8.524243e-13
BIN     1.906959e-13
BPM     2.221003e-13
BPVN    5.041352e-13
BNL     2.071852e-13
BPU     3.242469e-13
BUL     4.278564e-13
CAP     2.051893e-13
EDN     4.069630e-13
ENEL    4.965724e-14
ENI     1.376742e-13
FWB     2.257069e-13
F       2.514147e-13
FNC     1.000000e+00
G       5.820691e-13
ES      2.510243e-13
dtype: float64
Portfolio Variance: 0.00536394

Results for P_mu_2:
mu value: 0.00131479
P_mu_2:
AL      1.179575e-01
AGL     1.938823e-09
AUTO    3.463097e-09
NTV     6.121983e-09
BFI     2.119762e-01
BIN     5.084289e-09
BPM     5.772302e-09
```

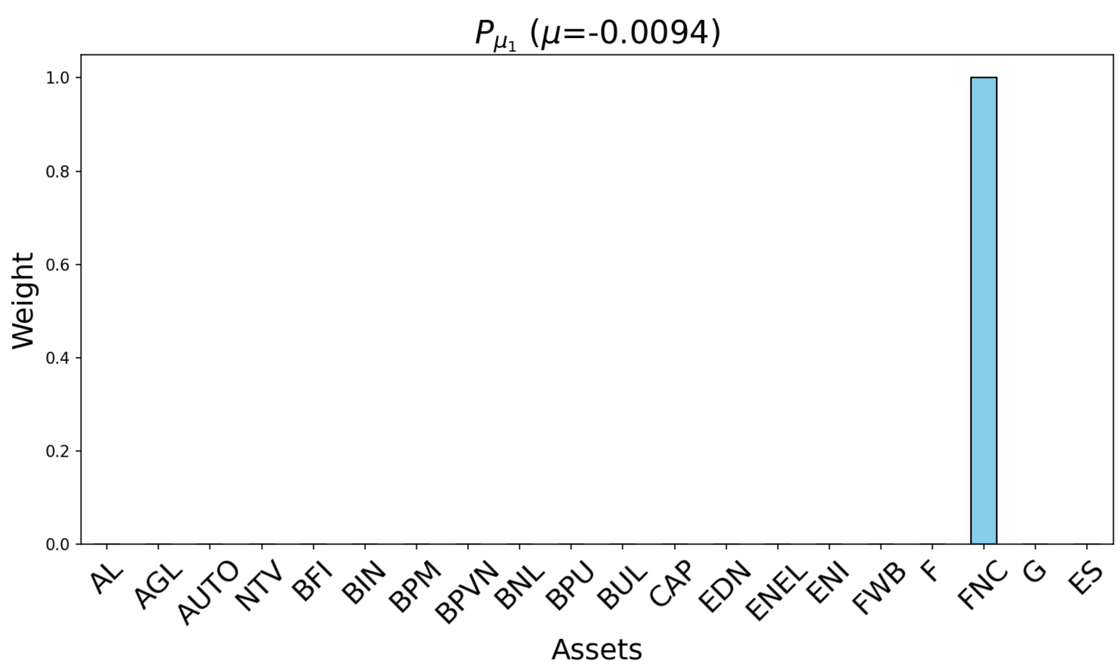
```
BPVN    1.354719e-01
BNL     3.889029e-09
BPU     7.645199e-09
BUL     2.499473e-01
CAP     6.346151e-09
EDN     1.010049e-08
ENEL    3.185919e-02
ENI     3.212012e-09
FWB     4.433937e-09
F       6.222333e-03
FNC     4.095426e-08
G       1.435122e-01
ES      1.030533e-01
dtype: float64
Portfolio Variance: 0.00084723

Results for P_mu_3:
mu value: 0.01203845
P_mu_3:
AL      3.937131e-02
AGL     4.589783e-09
AUTO    7.188299e-09
NTV     1.329231e-08
BFI     3.936444e-02
BIN     9.821857e-02
BPM     1.379199e-08
BPVN    5.582464e-08
BNL     9.880884e-09
BPU     8.114386e-09
BUL     6.407331e-02
CAP     1.390845e-02
EDN     1.194561e-08
ENEL    9.249308e-02
ENI     4.663966e-04
FWB     1.130931e-08
F       3.487857e-01
FNC     1.256211e-08
G       1.056945e-07
ES      3.033185e-01
dtype: float64
Portfolio Variance: 0.00053037

Results for P_mu_4:
mu value: 0.02276211
P_mu_4:
AL      3.025714e-09
AGL     1.202866e-08
AUTO    5.161618e-09
```

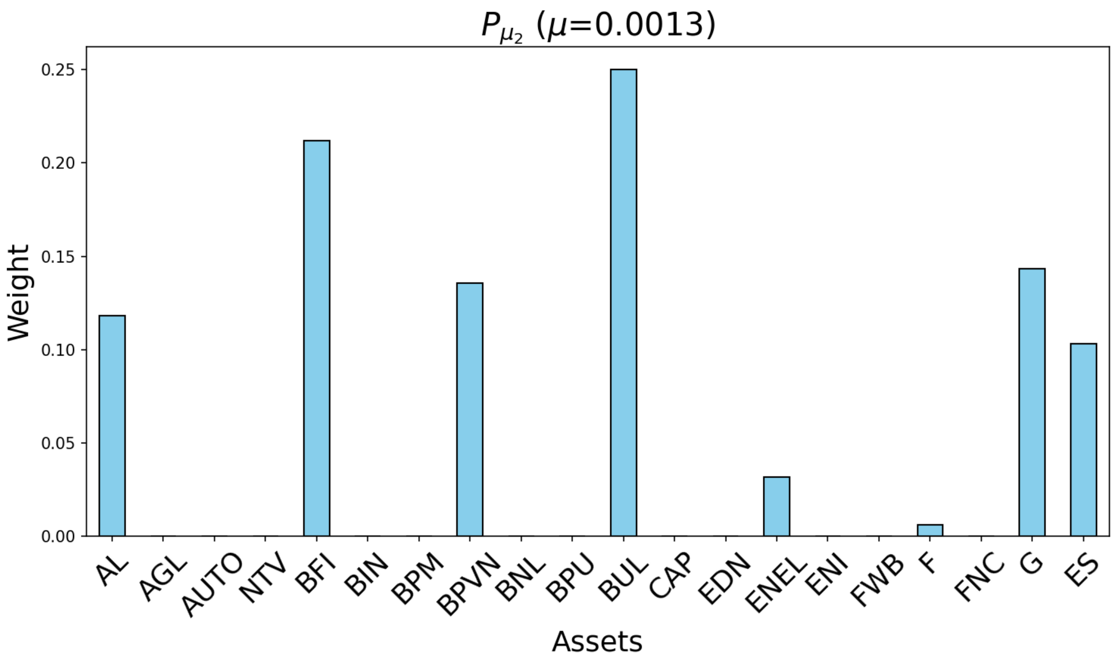
```
NTV     1.199812e-08
BFI     1.921227e-09
BIN     2.661385e-01
BPM     8.474513e-09
BPVN    2.381593e-09
BNL     8.255607e-08
BPU     2.066483e-09
BUL     2.468460e-09
CAP     2.321323e-01
EDN     3.607961e-09
ENEL    2.141669e-02
ENI     2.657369e-01
FWB     6.159767e-07
F       9.313679e-02
FNC     1.398843e-09
G       3.519287e-09
ES      1.214381e-01
dtype: float64
Portfolio Variance: 0.00095881

Results for P_mu_5:
mu value: 0.03348577
P_mu_5:
AL      5.930311e-14
AGL     5.603395e-12
AUTO    3.583971e-14
NTV     3.533156e-14
BFI     1.359417e-13
BIN     1.208569e-13
BPM     5.604470e-14
BPVN    1.252175e-13
BNL     8.965990e-14
BPU     9.746256e-14
BUL     1.193504e-13
CAP     8.304294e-14
EDN     1.191814e-13
ENEL    7.411095e-14
ENI     1.000000e+00
FWB     5.091547e-14
F       3.563191e-14
FNC     1.315366e-13
G       1.337141e-13
ES      3.529314e-14
dtype: float64
Portfolio Variance: 0.00444219
```

[19]:

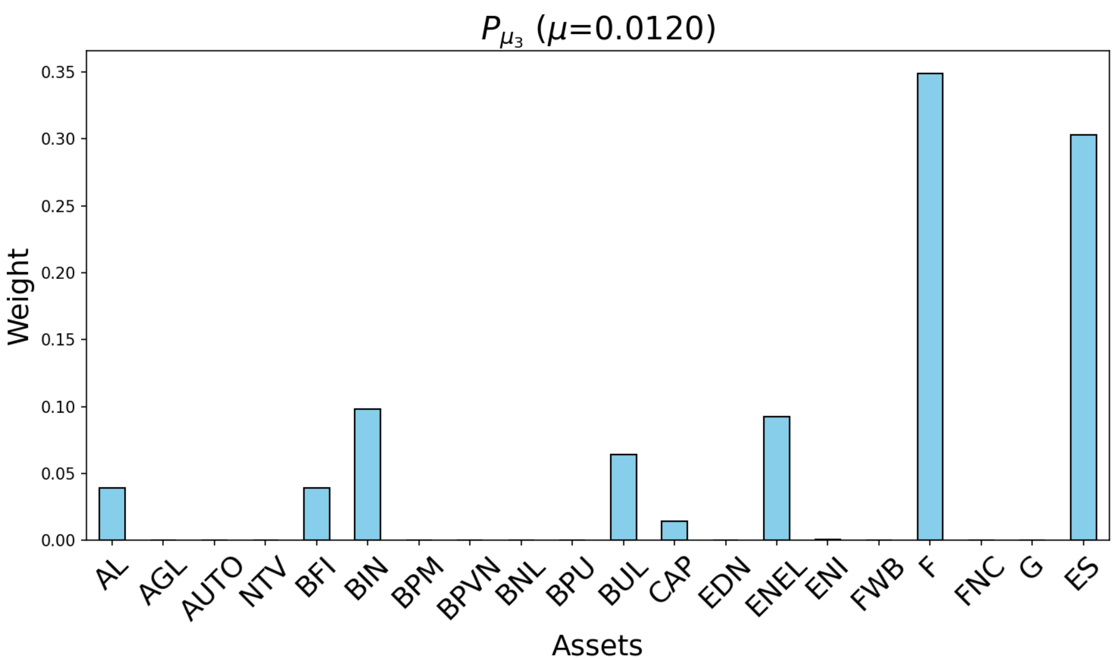
```
# Plotting the results
for key, result in m3_results.items():
    if isinstance(result, dict):
        m3_weights = result["m3_weights"]
        mu_value = result["mu_value"]

        # Create a bar plot for the weights
        plt.figure(figsize=(10, 6), dpi=150)
        m3_weights.plot(kind="bar", color="skyblue", edgecolor="black")
        plt.title(f"$P_{{\\mu_{key[-1]}}}$ ($\mu$={result['mu_value']:.4f})",␣
 ↪fontsize=20)
        plt.xlabel("Assets", fontsize=18)
        plt.ylabel("Weight", fontsize=18)
        plt.xticks(rotation=45, fontsize=18)
        plt.tight_layout()

        # Save the plot
        # plt.savefig(f"P_mu{key[-1]}_weights.png")
        plt.show()
```

```
<>:10: SyntaxWarning: invalid escape sequence '\m'
<>:10: SyntaxWarning: invalid escape sequence '\m'
/tmp/ipykernel_5831/1328220938.py:10: SyntaxWarning: invalid escape sequence
'\m'
  plt.title(f"$P_{{\\mu_{key[-1]}}}$ ($\mu$={result['mu_value']:.4f})",
fontsize=20)
```

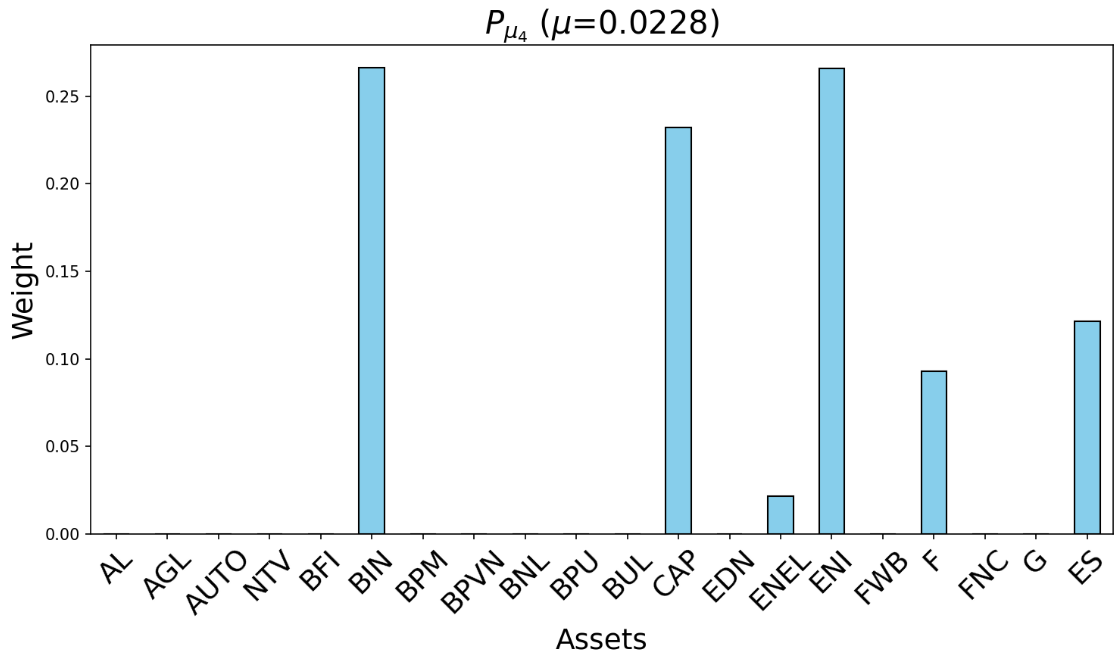


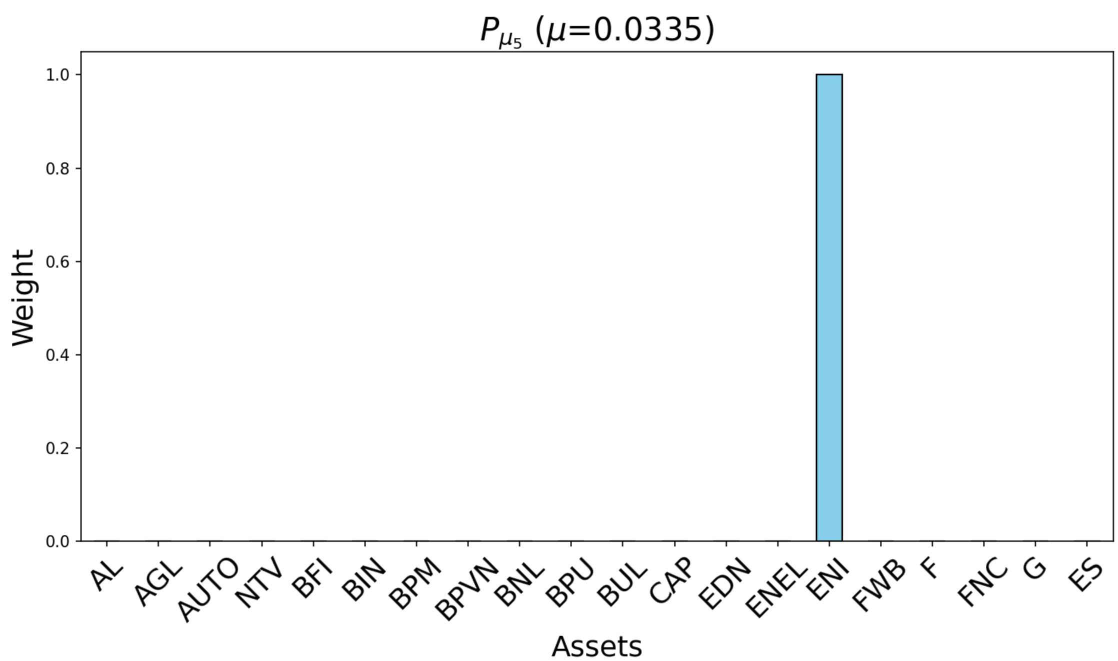


P_{μ_2} (μ=0.0013)
Weight
0.25
0.20
0.15
0.10
0.05
0.00
AL AGL AUTO NTV BFI BIN BPM BPVN BNL BPU BUL CAP EDN ENEL ENI FWB F FNC G ES
Assets



P_{μ_3} (μ=0.0120)
Weight
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
AL AGL AUTO NTV BFI BIN BPM BPVN BNL BPU BUL CAP EDN ENEL ENI FWB F FNC G ES
Assets





2.2 Exercise 2.3

- For each possible value $\mu_1, \mu_2, \mu_3, \mu_4, \mu_5$, solve model (MwV) with the additional constraint formulated in 1.2 by fixing $k = 2$. Denote by $PC_{\mu_1}, PC_{\mu_2}, PC_{\mu_3}, PC_{\mu_4}, PC_{\mu_5}$ the optimal portfolios so found.

[20]:
```
# Define the list of mu values
mu_values = [mu1, mu2, mu3, mu4, mu5]

# Create a dictionary to store results
m4_results = {}

# List to store data for plotting the efficient frontier
m4_efficient_frontier = []

#fix the cardinality constraint
k = 2

for i, mu in enumerate(mu_values, 1):
    # Create a new model
    m4 = gb.Model()

    # Add variables for portfolio weights
    x = pd.Series(m4.addVars(stocks, lb=0, name='X'), index=stocks)

    # Introduce binary variables to indicate if a stock is selected
    y = pd.Series(m4.addVars(stocks, vtype=gb.GRB.BINARY, name='Y'),␣
 ↪index=stocks)

    # Compute portfolio variance and return
    portfolio_variance = covariance_matrix.dot(x).dot(x)
    portfolio_return = expected_returns.dot(x)

    # Add objective function: Minimize portfolio variance
    m4.setObjective(portfolio_variance, sense=gb.GRB.MINIMIZE)

    # Add budget constraint: Sum of weights equals 1
    m4.addConstr(x.sum() == 1, name="budget")

    # Add portfolio return constraint
    m4_trc = m4.addConstr(portfolio_return == mu, name="return_constraint")

    # Add cardinality constraint
    for stock in stocks:
        cardinality_constraint = m4.addConstr(x[stock] <= y[stock],␣
 ↪name=f"selection_{stock}")
    m4.addConstr(y.sum() == k, name="cardinality_constraint")

    # Optimize model
    m4.optimize()
```

```python
    # Store results if feasible solution found
    if m4.status == gb.GRB.OPTIMAL:
        m4_weights = pd.Series({stock: x[stock].X for stock in stocks})
        m4_portfolio_variance_value = portfolio_variance.getValue()

        # Store results in a dictionary
        m4_results[f"PC_mu_{i}"] = {
            "mu_value": mu,
            "m4_weights": m4_weights,
            "m4_portfolio_variance": m4_portfolio_variance_value,
        }

        # Add data to the efficient frontier list
        m4_efficient_frontier.append({
            "mu": mu,
            "m4_variance": m4_portfolio_variance_value,
            "m4_weights": m4_weights,
        })
    else:
        m4_results[f"PC_mu_{i}"] = "No feasible solution found"

# Display results
for key, result in m4_results.items():
    if isinstance(result, dict):
        print(f"\nResults for {key}:")
        print(f"mu value: {result['mu_value']:.8f}")
        print(f"{key}:")
        print(result['m4_weights'])
        print(f"Portfolio Variance: {result['m4_portfolio_variance']:.8f}")
    else:
        print(f"\nResults for {key}: {result}")
```

```
Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (linux64 - "Ubuntu 20.04.6
LTS")

CPU model: Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 23 rows, 40 columns and 100 nonzeros
Model fingerprint: 0xf8da3012
Model has 210 quadratic objective terms
Variable types: 20 continuous, 20 integer (20 binary)
Coefficient statistics:
  Matrix range     [3e-04, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [3e-05, 1e-02]
```

```
  Bounds range     [1e+00, 1e+00]
  RHS range        [9e-03, 2e+00]
Found heuristic solution: objective 0.0053638
Presolve removed 1 rows and 1 columns
Presolve time: 0.00s
Presolved: 22 rows, 39 columns, 92 nonzeros
Presolved model has 210 quadratic objective terms
Found heuristic solution: objective 0.0053638
Variable types: 20 continuous, 19 integer (19 binary)

Explored 0 nodes (0 simplex iterations) in 0.01 seconds (0.00 work units)
Thread count was 2 (of 2 available processors)

Solution count 2: 0.00536383 0.00536384

Optimal solution found (tolerance 1.00e-04)
Best objective 5.363832675351e-03, best bound 5.363806018794e-03, gap 0.0005%
Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (linux64 - "Ubuntu 20.04.6
LTS")

CPU model: Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 23 rows, 40 columns and 100 nonzeros
Model fingerprint: 0x662efc6a
Model has 210 quadratic objective terms
Variable types: 20 continuous, 20 integer (20 binary)
Coefficient statistics:
  Matrix range     [3e-04, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [3e-05, 1e-02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e-03, 2e+00]
Presolve time: 0.00s
Presolved: 23 rows, 40 columns, 100 nonzeros
Presolved model has 210 quadratic objective terms
Variable types: 20 continuous, 20 integer (20 binary)
Found heuristic solution: objective 0.0040509
Found heuristic solution: objective 0.0017471

Root relaxation: objective 8.472343e-04, 33 iterations, 0.00 seconds (0.00 work
units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    0.00085    0    7    0.00175    0.00085  51.5%     -    0s
```

```
H    0     0                          0.0013944    0.00085  39.2%     -    0s
     0     0    0.00085    0    8     0.00139    0.00085  39.2%     -    0s
H    0     0                          0.0012898    0.00085  34.3%     -    0s
H    0     0                          0.0012891    0.00085  34.3%     -    0s
     0     0    0.00085    0    8     0.00129    0.00085  34.3%     -    0s
     0     0    0.00085    0    8     0.00129    0.00085  34.3%     -    0s
     0     0    0.00085    0    8     0.00129    0.00085  34.3%     -    0s
     0     2    0.00085    0    8     0.00129    0.00085  34.3%     -    0s
H    9     7                          0.0012743    0.00090  29.6%   7.3    0s
H   27    16                          0.0012699    0.00091  28.3%   7.3    0s

Cutting planes:
  Cover: 1
  Implied bound: 3
  MIR: 3
  Flow cover: 7
  Inf proof: 1

Explored 79 nodes (536 simplex iterations) in 0.06 seconds (0.01 work units)
Thread count was 2 (of 2 available processors)

Solution count 7: 0.00126995 0.00127432 0.00128912 … 0.00405089

Optimal solution found (tolerance 1.00e-04)
Best objective 1.269949518138e-03, best bound 1.269949518138e-03, gap 0.0000%
Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (linux64 - "Ubuntu 20.04.6
LTS")

CPU model: Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 23 rows, 40 columns and 100 nonzeros
Model fingerprint: 0x5668d0fc
Model has 210 quadratic objective terms
Variable types: 20 continuous, 20 integer (20 binary)
Coefficient statistics:
  Matrix range     [3e-04, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [3e-05, 1e-02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e-02, 2e+00]
Presolve time: 0.00s
Presolved: 23 rows, 40 columns, 100 nonzeros
Presolved model has 210 quadratic objective terms
Variable types: 20 continuous, 20 integer (20 binary)
Found heuristic solution: objective 0.0030692
Found heuristic solution: objective 0.0014375
```

```
Root relaxation: objective 5.303670e-04, 34 iterations, 0.00 seconds (0.00 work
units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    0.00053    0    8    0.00144    0.00053  63.1%     -    0s
H    0     0                       0.0011141    0.00053  52.4%     -    0s
     0     0    0.00053    0    9    0.00111    0.00053  52.4%     -    0s
     0     0    0.00053    0    9    0.00111    0.00053  52.4%     -    0s
     0     0    0.00053    0    9    0.00111    0.00053  52.4%     -    0s
     0     2    0.00053    0    9    0.00111    0.00053  52.1%     -    0s

Cutting planes:
  Cover: 1
  Implied bound: 4
  MIR: 1
  Flow cover: 3

Explored 55 nodes (435 simplex iterations) in 0.06 seconds (0.01 work units)
Thread count was 2 (of 2 available processors)

Solution count 3: 0.00111406 0.00143747 0.00306919

Optimal solution found (tolerance 1.00e-04)
Best objective 1.114063779235e-03, best bound 1.114063779235e-03, gap 0.0000%
Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (linux64 - "Ubuntu 20.04.6
LTS")

CPU model: Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 23 rows, 40 columns and 100 nonzeros
Model fingerprint: 0x5f7a278d
Model has 210 quadratic objective terms
Variable types: 20 continuous, 20 integer (20 binary)
Coefficient statistics:
  Matrix range     [3e-04, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [3e-05, 1e-02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [2e-02, 2e+00]
Presolve time: 0.00s
Presolved: 23 rows, 40 columns, 100 nonzeros
Presolved model has 210 quadratic objective terms
Variable types: 20 continuous, 20 integer (20 binary)
```

```
Found heuristic solution: objective 0.0041199
Found heuristic solution: objective 0.0015454

Root relaxation: objective 9.588071e-04, 37 iterations, 0.00 seconds (0.00 work
units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    0.00096    0    6    0.00155    0.00096  38.0%     -    0s
     0     0    0.00096    0    7    0.00155    0.00096  38.0%     -    0s
     0     0    0.00096    0    7    0.00155    0.00096  38.0%     -    0s
     0     0    0.00096    0    7    0.00155    0.00096  38.0%     -    0s
     0     0    0.00096    0    7    0.00155    0.00096  38.0%     -    0s
     0     0    0.00096    0    7    0.00155    0.00096  38.0%     -    0s
     0     2    0.00096    0    7    0.00155    0.00096  38.0%     -    0s

Cutting planes:
  Cover: 1
  Implied bound: 2
  MIR: 1
  Flow cover: 1

Explored 40 nodes (284 simplex iterations) in 0.04 seconds (0.00 work units)
Thread count was 2 (of 2 available processors)

Solution count 2: 0.00154535 0.00411986

Optimal solution found (tolerance 1.00e-04)
Best objective 1.545352117504e-03, best bound 1.545352117504e-03, gap 0.0000%
Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (linux64 - "Ubuntu 20.04.6
LTS")

CPU model: Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 23 rows, 40 columns and 100 nonzeros
Model fingerprint: 0x1018e578
Model has 210 quadratic objective terms
Variable types: 20 continuous, 20 integer (20 binary)
Coefficient statistics:
  Matrix range     [3e-04, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [3e-05, 1e-02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [3e-02, 2e+00]
Found heuristic solution: objective 0.0044418
```

```
Presolve time: 0.00s
Presolved: 23 rows, 40 columns, 100 nonzeros
Presolved model has 210 quadratic objective terms
Variable types: 20 continuous, 20 integer (20 binary)

Root relaxation: cutoff, 38 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0     cutoff    0         0.00444    0.00444  0.00%     -    0s

Explored 1 nodes (38 simplex iterations) in 0.01 seconds (0.00 work units)
Thread count was 2 (of 2 available processors)

Solution count 1: 0.00444181

Optimal solution found (tolerance 1.00e-04)
Best objective 4.441806210893e-03, best bound 4.441806210893e-03, gap 0.0000%

Results for PC_mu_1:
mu value: -0.00940887
PC_mu_1:
AL      0.000000
AGL     0.000000
AUTO    0.000000
NTV     0.000000
BFI     0.000000
BIN     0.000000
BPM     0.000000
BPVN    0.000000
BNL     0.000000
BPU     0.000000
BUL     0.000000
CAP     0.000000
EDN     0.000000
ENEL    0.000000
ENI     0.000000
FWB     0.000000
F       0.000000
FNC     0.999986
G       0.000014
ES      0.000000
dtype: float64
Portfolio Variance: 0.00536383

Results for PC_mu_2:
mu value: 0.00131479
```

```
PC_mu_2:
AL      0.000000
AGL     0.000000
AUTO    0.000000
NTV     0.000000
BFI     0.000000
BIN     0.000000
BPM     0.000000
BPVN    0.000000
BNL     0.000000
BPU     0.000000
BUL     0.956046
CAP     0.000000
EDN     0.000000
ENEL    0.043954
ENI     0.000000
FWB     0.000000
F       0.000000
FNC     0.000000
G       0.000000
ES      0.000000
dtype: float64
Portfolio Variance: 0.00126995

Results for PC_mu_3:
mu value: 0.01203845
PC_mu_3:
AL      0.075899
AGL     0.000000
AUTO    0.000000
NTV     0.000000
BFI     0.000000
BIN     0.000000
BPM     0.000000
BPVN    0.000000
BNL     0.000000
BPU     0.000000
BUL     0.000000
CAP     0.000000
EDN     0.000000
ENEL    0.000000
ENI     0.000000
FWB     0.000000
F       0.924101
FNC     0.000000
G       0.000000
ES      0.000000
dtype: float64
```

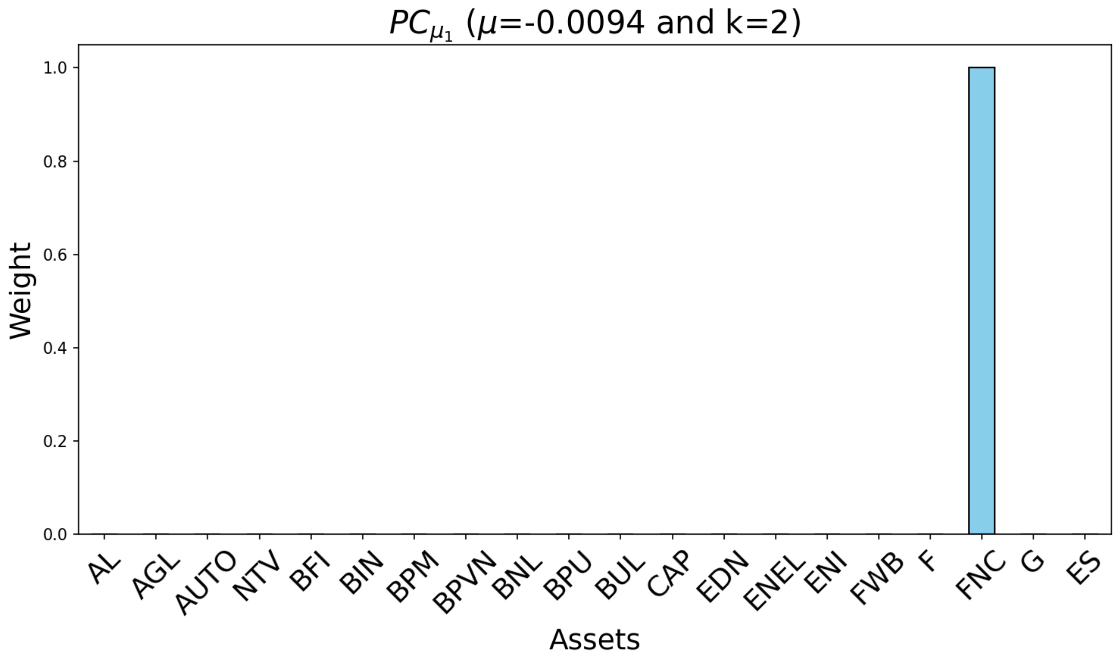
```
Portfolio Variance: 0.00111406

Results for PC_mu_4:
mu value: 0.02276211
PC_mu_4:
AL      0.000000
AGL     0.000000
AUTO    0.000000
NTV     0.000000
BFI     0.000000
BIN     0.000000
BPM     0.000000
BPVN    0.000000
BNL     0.000000
BPU     0.000000
BUL     0.000000
CAP     0.000000
EDN     0.000000
ENEL    0.000000
ENI     0.478648
FWB     0.000000
F       0.000000
FNC     0.000000
G       0.000000
ES      0.521352
dtype: float64
Portfolio Variance: 0.00154535

Results for PC_mu_5:
mu value: 0.03348577
PC_mu_5:
AL      0.000000
AGL     0.000000
AUTO    0.000000
NTV     0.000000
BFI     0.000000
BIN     0.000000
BPM     0.000054
BPVN    0.000000
BNL     0.000000
BPU     0.000000
BUL     0.000000
CAP     0.000000
EDN     0.000000
ENEL    0.000000
ENI     0.999946
FWB     0.000000
F       0.000000
```

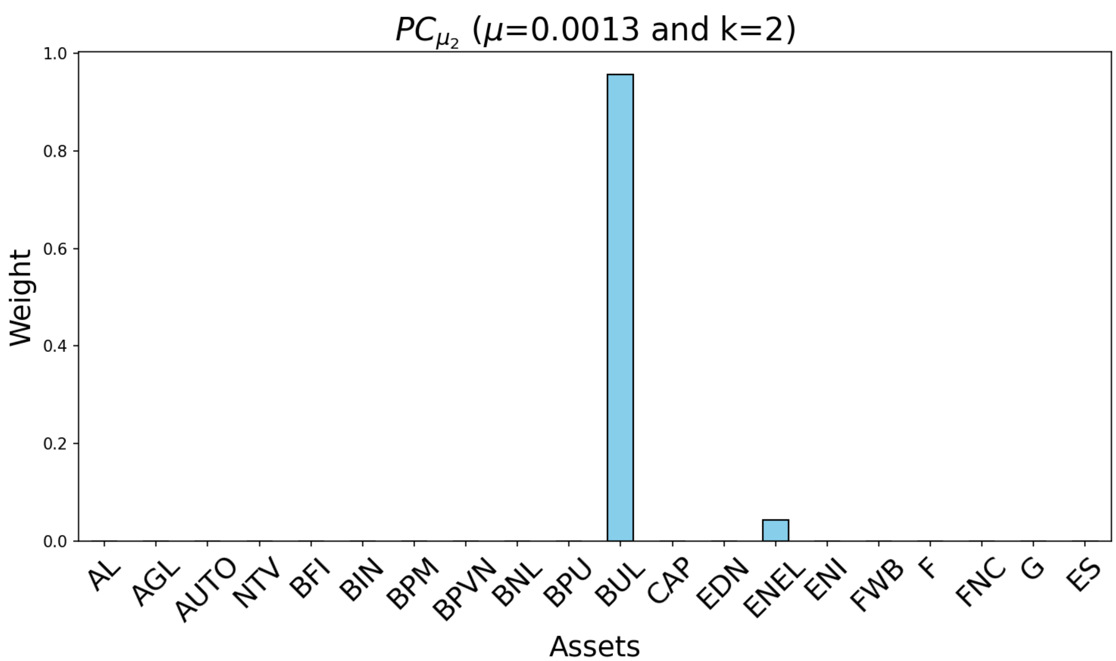
```
FNC      0.000000
G        0.000000
ES       0.000000
dtype: float64
Portfolio Variance: 0.00444181
```

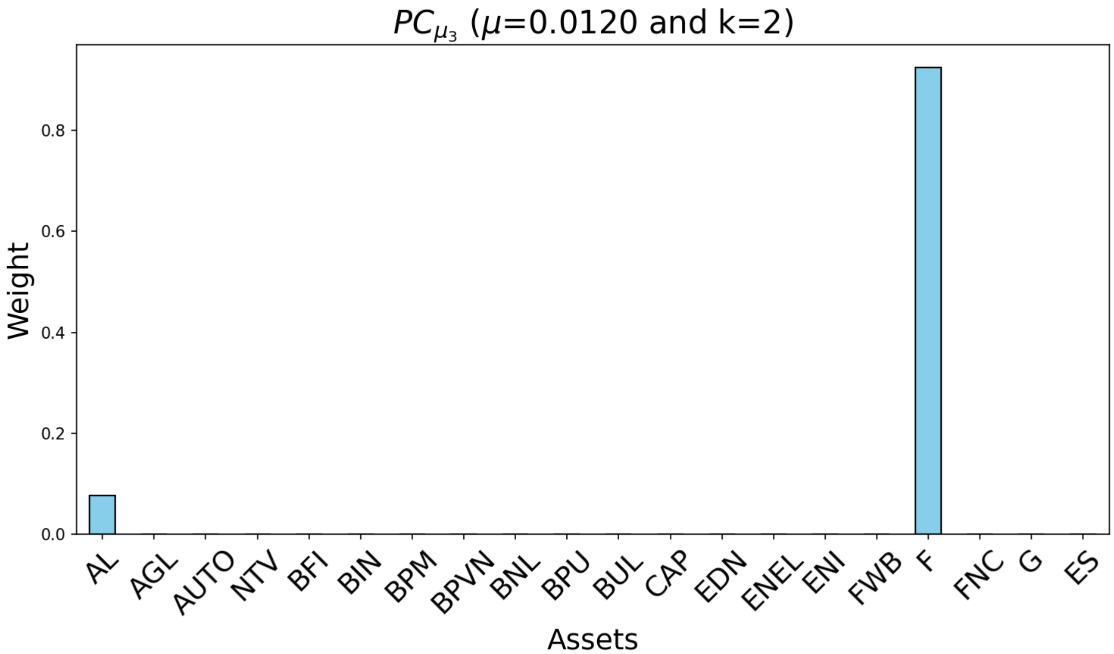
[21]:

```
# Plotting the results
for key, result in m4_results.items():
    if isinstance(result, dict):
        m4_weights = result["m4_weights"]
        mu_value = result["mu_value"]

        # Create a bar plot for the weights
        plt.figure(figsize=(10, 6), dpi=150)
        m4_weights.plot(kind="bar", color="skyblue", edgecolor="black")
        plt.title(f"$PC_{{\\mu_{key[-1]}}}$ ($\mu$={result['mu_value']:.4f} and␣
↪k=2)", fontsize=20)
        plt.xlabel("Assets", fontsize=18)
        plt.ylabel("Weight", fontsize=18)
        plt.xticks(rotation=45, fontsize=18)
        plt.tight_layout()

        # Save the plot
        # plt.savefig(f"PC_mu{key[-1]}_weights.png")
        plt.show()
```

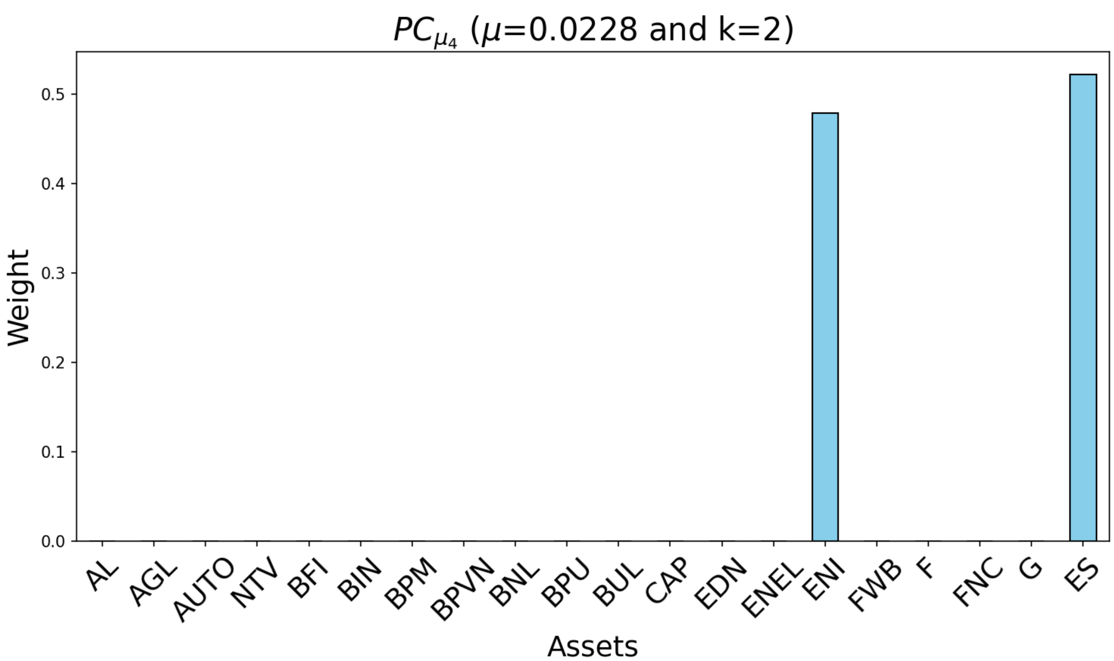
```
<>:10: SyntaxWarning: invalid escape sequence '\m'
<>:10: SyntaxWarning: invalid escape sequence '\m'
/tmp/ipykernel_5831/161338576.py:10: SyntaxWarning: invalid escape sequence '\m'
  plt.title(f"$PC_{{\\mu_{key[-1]}}}$ ($\mu$={result['mu_value']:.4f} and k=2)",
fontsize=20)
```


PC_{μ_1} (μ=-0.0094 and k=2)
Weight
1.0
0.8
0.6
0.4
0.2
0.0
AL AGL AUTO NTV BFI BIN BPM BPVN BNL BPU BUL CAP EDN ENEL ENI FWB F FNC G ES
Assets

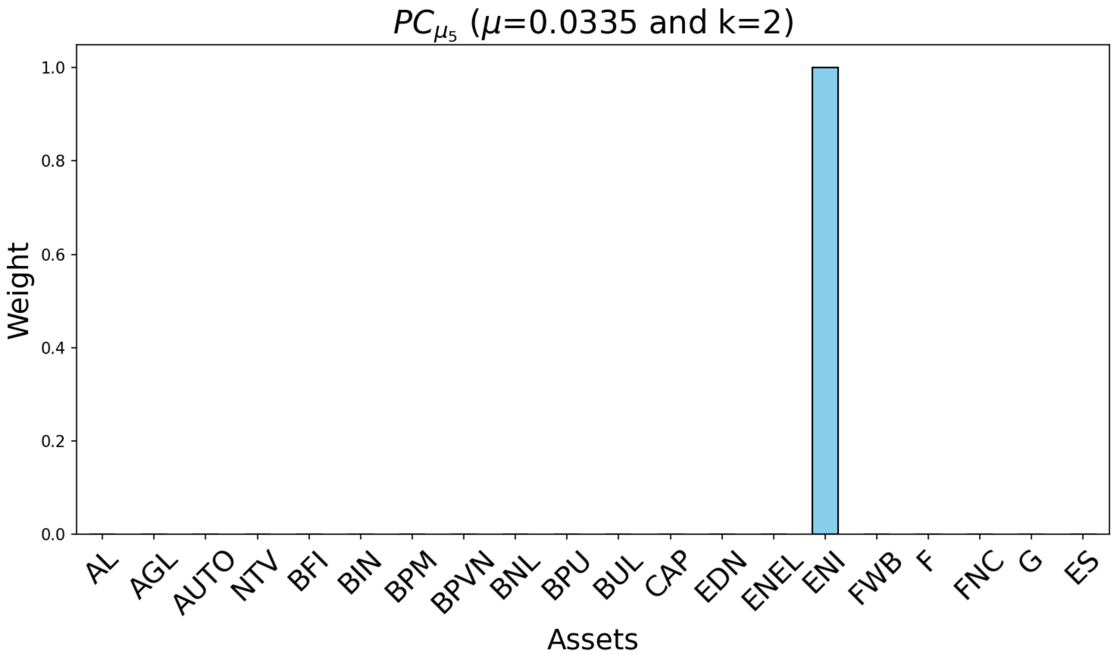

PC_{μ_2} (μ=0.0013 and k=2)
Weight
1.0
0.8
0.6
0.4
0.2
0.0
AL AGL AUTO NTV BFI BIN BPM BPVN BNL BPU BUL CAP EDN ENEL ENI FWB F FNC G ES
Assets



PC_{μ_3} (μ=0.0120 and k=2)
Weight
0.0
0.2
0.4
0.6
0.8
AL AGL AUTO NTV BFI BIN BPM BPVN BNL BPU BUL CAP EDN ENEL ENI FWB F FNC G ES
Assets



PC_{μ_4} (μ=0.0228 and k=2)
Weight
0.0
0.1
0.2
0.3
0.4
0.5
AL AGL AUTO NTV BFI BIN BPM BPVN BNL BPU BUL CAP EDN ENEL ENI FWB F FNC G ES
Assets



[22]:
```python
# Extract data for plotting
m4_mu_values_plot = [entry["mu"] for entry in m4_efficient_frontier]
m4_variances_plot = [entry["m4_variance"] for entry in m4_efficient_frontier]
m3_mu_values_plot = [entry["mu"] for entry in m3_efficient_frontier]
m3_variances_plot = [entry["m3_variance"] for entry in m3_efficient_frontier]

stock_stds = df[stocks].std()
stock_returns = df[stocks].mean()


plt.figure(figsize=(10, 6), dpi=150)
plt.plot(m4_variances_plot, m4_mu_values_plot, marker="o", color='red',␣
 ↪label="Efficient Frontier ($PC_µ$)")
plt.plot(m3_variances_plot, m3_mu_values_plot, marker="o", color='blue',␣
 ↪label="Efficient Frontier ($P_µ$)")
# Visualize stocks

plt.xlabel("Standard Deviation ($\sigma$)", fontsize=18)
plt.ylabel("Expected Return ($\mu$)", fontsize=18)
plt.title("Efficient Frontier ($PC_µ$ and $P_µ$)", fontsize=20)
plt.legend()
plt.grid()
plt.show()
```

```
<>:16: SyntaxWarning: invalid escape sequence '\s'
<>:17: SyntaxWarning: invalid escape sequence '\m'
```

```
<>:16: SyntaxWarning: invalid escape sequence '\s'
<>:17: SyntaxWarning: invalid escape sequence '\m'
/tmp/ipykernel_5831/49823176.py:16: SyntaxWarning: invalid escape sequence '\s'
  plt.xlabel("Standard Deviation ($\sigma$)", fontsize=18)
/tmp/ipykernel_5831/49823176.py:17: SyntaxWarning: invalid escape sequence '\m'
  plt.ylabel("Expected Return ($\mu$)", fontsize=18)
```