MScFE 622 STOCHASTIC MODELING

Group Work Project: 3

Group: 7372

Group Members:

- Gabriel Omondi Odero
- Eric Komla Anku
- Alpha K Kamara

Step 1: Literature Review and Selective Analysis of Huo's Multi-Armed Bandit Paper

The paper investigates the risk-aware multi-armed bandit problem, a framework for sequential decision-making under uncertainty. Huo's Risk-aware multi-armed bandit problem with application to portfolio selection outlines key points on using multi-armed bandits for portfolio selection. The goal is to balance trying new investments and sticking with existing profitable ones. A key part is Algorithm 1, which shows the steps for applying reinforcement learning to guide portfolio choices over time. It also brings risk measures like CVaR, so downside potential factors into the decisions. While theoretical proofs aren't needed for this task, it's good to know risk drives the strategies.

Hou and Fu tested their ideas on 30 stocks from places like banks and other companies. We'll use that list of stocks later to get data to try the approach. The main sections to focus on cover how they set up the portfolio selection problem what data you need, and the details of Algorithm 1 to walk through implementing their ideas (Huo and Fu).

Step 2: Defining Portfolio Selection as a Multi-Armed Bandit Problem with Pseudocode

Pseudocode

1. Obtain historical asset returns for each asset at the current time to infinity.

2. Select a basket of assets using graph approach to filter those assets.

3. At each time, select a portfolio and observe the rewards.

Step 3: Data Collection and Preparation for Financial and Non-Financial Institutions

a. Member A collects the data for 15 financial institutions (JPM,WFC, BAC, C, GS, USB, MS, KEY, PNC, COF, AXP, PRU, SCHW, BBT, STI)

b. Member B collects the data for 15 non-financial institutions (KR, PFE, XOM, WMT, DAL, CSCO, HCP, EQIX, DUK, NFLX, GE, APA, F, REGN, CMS).

c. Member C combines the data into a suitable Python time series data structure. Member C will also compute the daily returns of all 30 series.

In [ ]:
```
# Import necessary libraries
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

# Set up date range for data retrieval
start_date = '2008-09-01'
end_date = '2008-10-31'

# Define the lists of financial and non-financial institutions
financial_institutions = ["JPM", "WFC", "BAC", "C", "GS", "USB", "MS", "KEY", "PNC"
non_financial_institutions = ["KR", "PFE", "XOM", "WMT", "DAL", "CSCO", "DOC", "EQI

# Function to get historical data from Yahoo Finance
def get_historical_data(symbols, start, end):
    data_dict = {}
    for symbol in symbols:
        try:
            stock_data = yf.Ticker(symbol).history(start=start, end=end)
            if not stock_data.empty:
                data_dict[symbol] = stock_data['Close']  # Collect only the closing
                print(f"Data for {symbol} retrieved successfully.")
            else:
                print(f"No data found for {symbol}.")
        except Exception as e:
            print(f"Error retrieving data for {symbol}: {e}")
    return pd.DataFrame(data_dict)

# Collect historical data for financial institutions
financial_data = get_historical_data(financial_institutions, start_date, end_date)

# Collect historical data for non-financial institutions
non_financial_data = get_historical_data(non_financial_institutions, start_date, en

# Merge data into a unified time series
data_combined = pd.concat([financial_data, non_financial_data], axis=1)
```

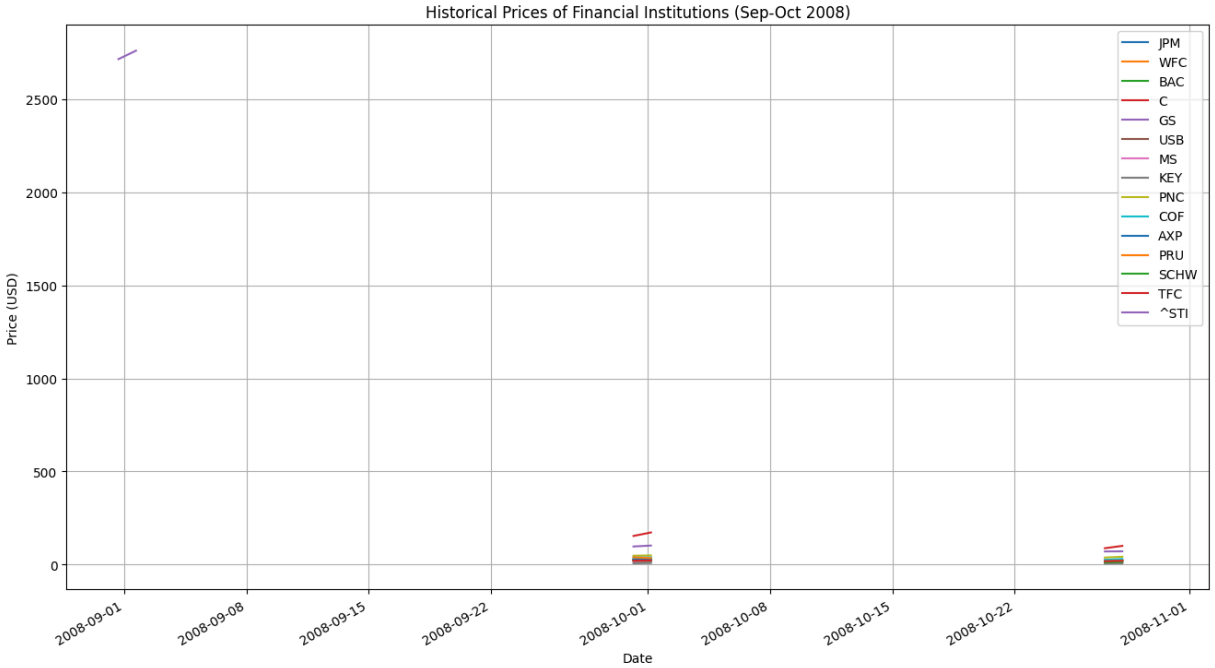
```
# Calculate daily returns for each company
data_combined_returns = data_combined.pct_change().dropna()

# Save the data to CSV files for reporting purposes
data_combined.to_csv("historical_prices_combined.csv")
data_combined_returns.to_csv("daily_returns_combined.csv")

# Display a sample of the data
print("Sample of historical prices:")
print(data_combined.head())

print("\nSample of daily returns:")
print(data_combined_returns.head())

# Plot historical prices for financial institutions
financial_data.plot(figsize=(16, 9), title="Historical Prices of Financial Institut
plt.xlabel("Date")
plt.ylabel("Price (USD)")
plt.legend(loc='upper right')
plt.show()

# Plot daily returns for financial institutions
financial_data_returns = financial_data.pct_change().dropna()
financial_data_returns.plot(figsize=(16, 9), title="Daily Returns of Financial Inst
plt.xlabel("Date")
plt.ylabel("Daily Return")
plt.legend(loc='upper right')
plt.show()

# Plot historical prices for non-financial institutions
non_financial_data.plot(figsize=(16, 9), title="Historical Prices of Non-Financial
plt.xlabel("Date")
plt.ylabel("Price (USD)")
plt.legend(loc='upper right')
plt.show()

# Plot daily returns for non-financial institutions
non_financial_data_returns = non_financial_data.pct_change().dropna()
non_financial_data_returns.plot(figsize=(16, 9), title="Daily Returns of Non-Financ
plt.xlabel("Date")
plt.ylabel("Daily Return")
plt.legend(loc='upper right')
plt.show()

# Display daily returns in a table format for reporting
print("\nTable of daily returns:")
print(data_combined_returns)
```

```
Data for JPM retrieved successfully.
Data for WFC retrieved successfully.
Data for BAC retrieved successfully.
Data for C retrieved successfully.
Data for GS retrieved successfully.
Data for USB retrieved successfully.
Data for MS retrieved successfully.
Data for KEY retrieved successfully.
Data for PNC retrieved successfully.
Data for COF retrieved successfully.
Data for AXP retrieved successfully.
Data for PRU retrieved successfully.
Data for SCHW retrieved successfully.
Data for TFC retrieved successfully.
Data for ^STI retrieved successfully.
Data for KR retrieved successfully.
Data for PFE retrieved successfully.
Data for XOM retrieved successfully.
Data for WMT retrieved successfully.
Data for DAL retrieved successfully.
Data for CSCO retrieved successfully.
Data for DOC retrieved successfully.
Data for EQIX retrieved successfully.
Data for DUK retrieved successfully.
Data for NFLX retrieved successfully.
Data for GE retrieved successfully.
Data for APA retrieved successfully.
Data for F retrieved successfully.
Data for REGN retrieved successfully.
Data for CMS retrieved successfully.
Sample of historical prices:
                                  JPM        WFC        BAC           C  \
Date
2008-08-31 16:00:00+00:00        NaN        NaN        NaN         NaN
2008-09-01 16:00:00+00:00        NaN        NaN        NaN         NaN
2008-09-02 04:00:00+00:00  25.863001  20.452135  24.837965  143.582718
2008-09-02 16:00:00+00:00        NaN        NaN        NaN         NaN
2008-09-03 04:00:00+00:00  26.340590  20.321085  25.591097  147.339417

                                   GS        USB         MS       KEY  \
Date
2008-08-31 16:00:00+00:00         NaN        NaN        NaN       NaN
2008-09-01 16:00:00+00:00         NaN        NaN        NaN       NaN
2008-09-02 04:00:00+00:00  126.012177  20.446829  29.067612  7.853099
2008-09-02 16:00:00+00:00         NaN        NaN        NaN       NaN
2008-09-03 04:00:00+00:00  127.757706  20.813185  29.679951  7.921660

                                  PNC        COF  ...       CSCO        DOC  \
Date                                              ...
2008-08-31 16:00:00+00:00        NaN        NaN  ...        NaN        NaN
2008-09-01 16:00:00+00:00        NaN        NaN  ...        NaN        NaN
2008-09-02 04:00:00+00:00  47.275421  34.334396  ...  15.969232  13.844852
2008-09-02 16:00:00+00:00        NaN        NaN  ...        NaN        NaN
2008-09-03 04:00:00+00:00  47.713188  34.900021  ...  15.673382  13.795016

                                 EQIX        DUK      NFLX         GE  \
```

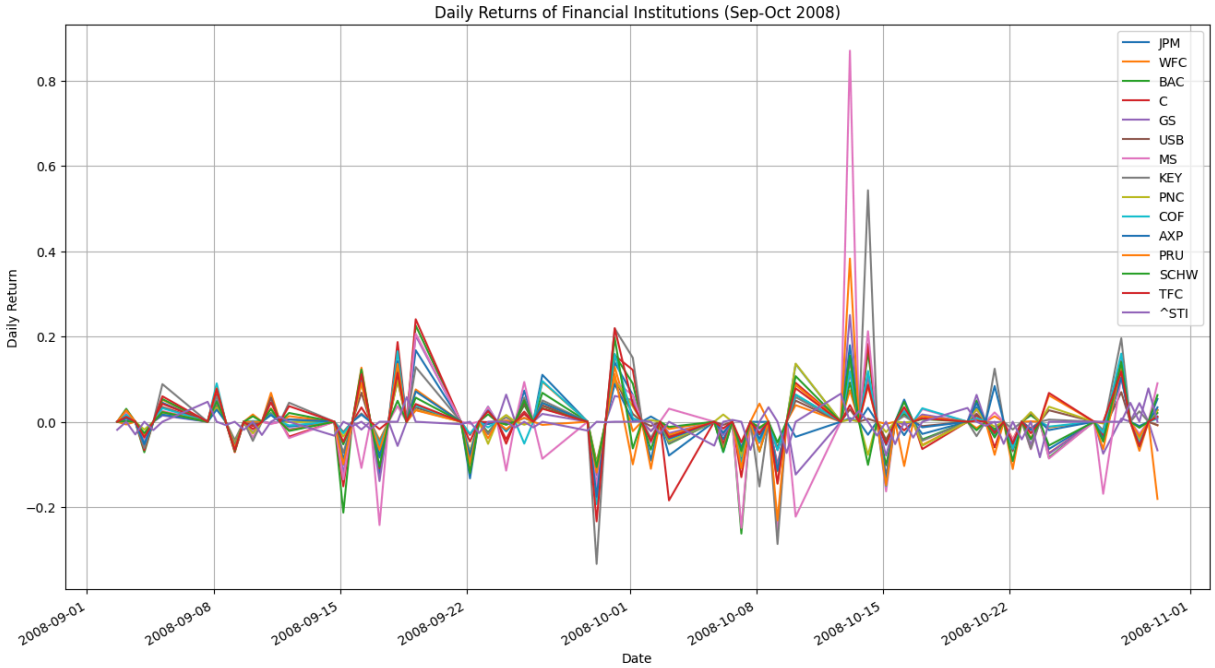
```
Date
2008-08-31 16:00:00+00:00        NaN        NaN       NaN        NaN
2008-09-01 16:00:00+00:00        NaN        NaN       NaN        NaN
2008-09-02 04:00:00+00:00  61.828243  25.112762  4.405714  91.865829
2008-09-02 16:00:00+00:00        NaN        NaN       NaN        NaN
2008-09-03 04:00:00+00:00  61.217213  24.794155  4.415714  91.994644

                                 APA         F       REGN       CMS
Date
2008-08-31 16:00:00+00:00        NaN       NaN        NaN       NaN
2008-09-01 16:00:00+00:00        NaN       NaN        NaN       NaN
2008-09-02 04:00:00+00:00  81.725029  2.525400  20.580000  7.950073
2008-09-02 16:00:00+00:00        NaN       NaN        NaN       NaN
2008-09-03 04:00:00+00:00  82.601143  2.558997  21.799999  7.856610

[5 rows x 30 columns]

Sample of daily returns:
                                 JPM       WFC       BAC         C        GS  \
Date
2008-09-02 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000
2008-09-03 04:00:00+00:00  0.018466 -0.006408  0.030322  0.026164  0.013852
2008-09-03 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000
2008-09-04 04:00:00+00:00 -0.045329 -0.043212 -0.071602 -0.066803 -0.040034
2008-09-04 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000

                                 USB        MS       KEY       PNC       COF  \
Date
2008-09-02 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000
2008-09-03 04:00:00+00:00  0.017917  0.021066  0.008731  0.009260  0.016474
2008-09-03 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000
2008-09-04 04:00:00+00:00 -0.039453 -0.043396 -0.062156 -0.018484 -0.051029
2008-09-04 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000

                           ...      CSCO       DOC      EQIX       DUK  \
Date                       ...
2008-09-02 16:00:00+00:00  ...  0.000000  0.000000  0.000000  0.000000
2008-09-03 04:00:00+00:00  ... -0.018526 -0.003600 -0.009883 -0.012687
2008-09-03 16:00:00+00:00  ...  0.000000  0.000000  0.000000  0.000000
2008-09-04 04:00:00+00:00  ... -0.044187 -0.040289 -0.033063  0.009930
2008-09-04 16:00:00+00:00  ...  0.000000  0.000000  0.000000  0.000000

                                 NFLX        GE       APA         F      REGN  \
Date
2008-09-02 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000
2008-09-03 04:00:00+00:00  0.002270  0.001402  0.010720  0.013304  0.059281
2008-09-03 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000
2008-09-04 04:00:00+00:00 -0.033646 -0.030451  0.023725 -0.039387 -0.065138
2008-09-04 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000

                                 CMS
Date
2008-09-02 16:00:00+00:00  0.000000
2008-09-03 04:00:00+00:00 -0.011756
2008-09-03 16:00:00+00:00  0.000000
2008-09-04 04:00:00+00:00  0.001488
```

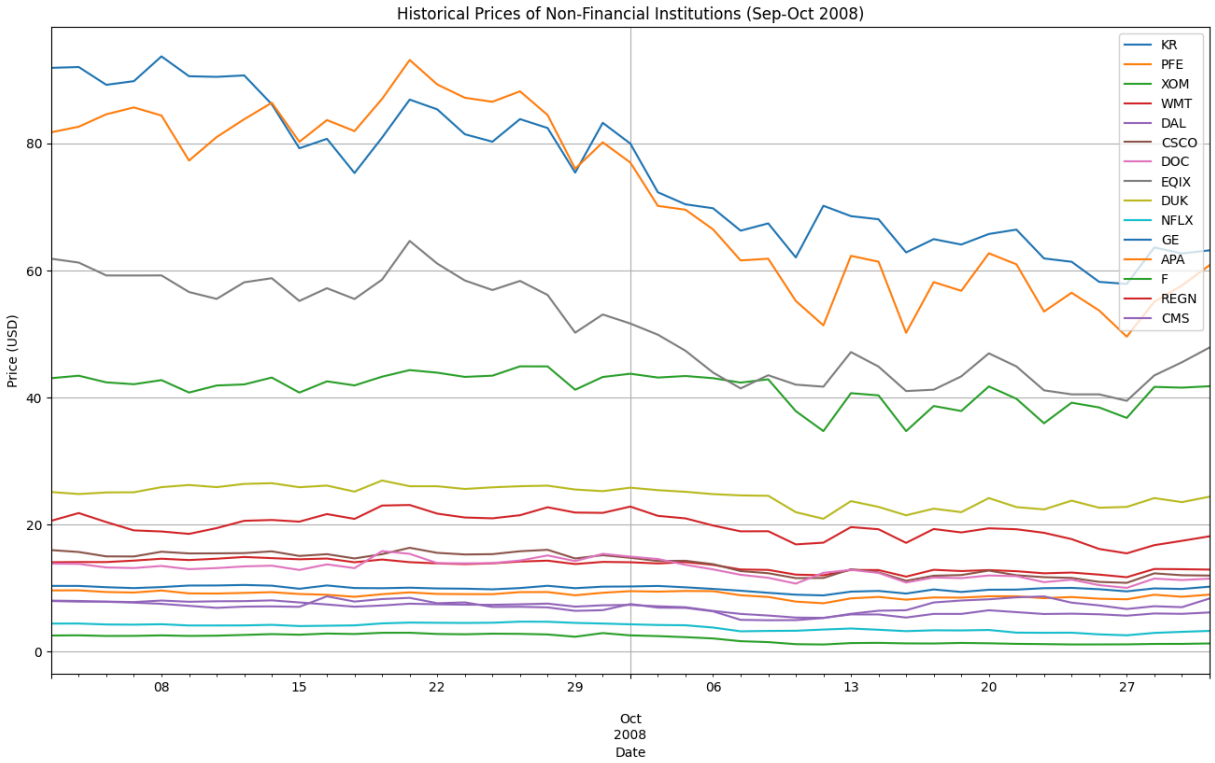
```
2008-09-04 16:00:00+00:00  0.000000

[5 rows x 30 columns]
```

```
<ipython-input-1-2950400199c1>:39: FutureWarning: The default fill_method='pad' in DataFrame.pct_change is deprecated and will be removed in a future version. Either fill in any non-leading NA values prior to calling pct_change or specify 'fill_method=None' to not fill NA values.
  data_combined_returns = data_combined.pct_change().dropna()
```




```
<ipython-input-1-2950400199c1>:60: FutureWarning: The default fill_method='pad' in DataFrame.pct_change is deprecated and will be removed in a future version. Either fill in any non-leading NA values prior to calling pct_change or specify 'fill_method=None' to not fill NA values.
  financial_data_returns = financial_data.pct_change().dropna()
```




Historical Prices of Non-Financial Institutions (Sep-Oct 2008)
Price (USD)
0
20
40
60
80
08
15
22
29
06
13
20
27
Oct
2008
Date
KR
PFE
XOM
WMT
DAL
CSCO
DOC
EQIX
DUK
NFLX
GE
APA
F
REGN
CMS

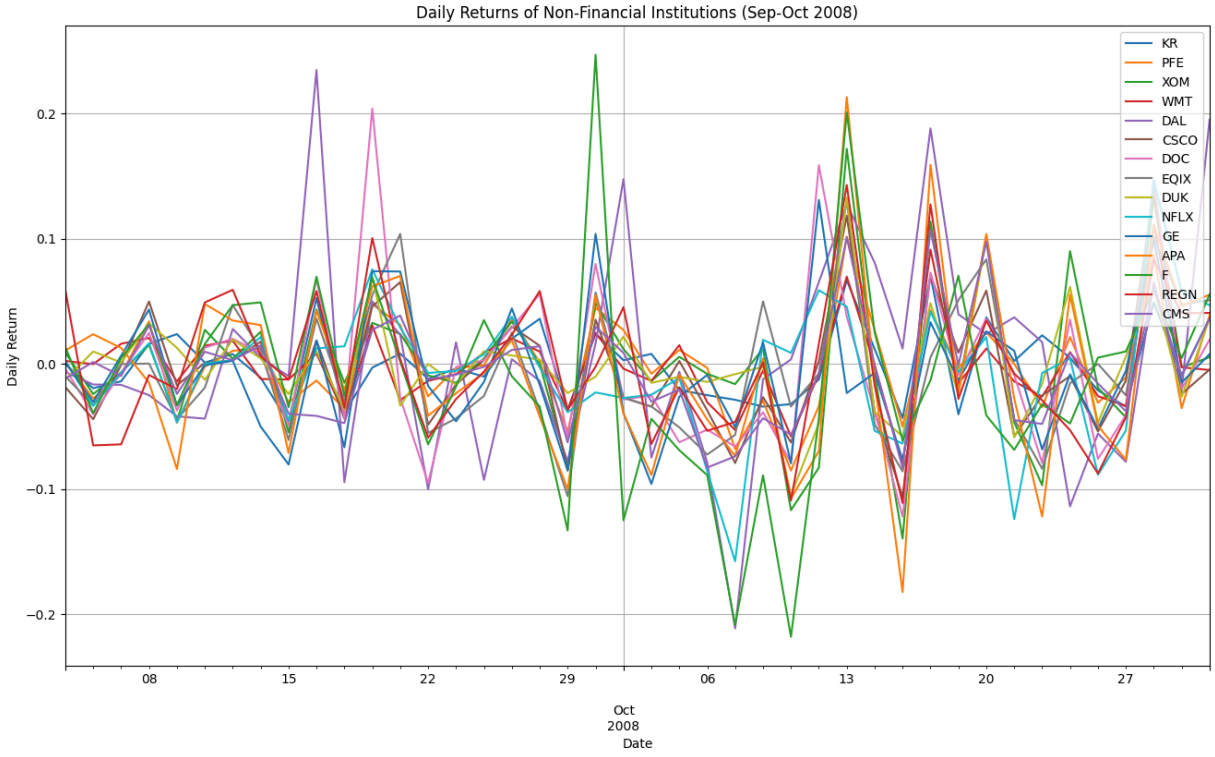

Daily Returns of Non-Financial Institutions (Sep-Oct 2008)
Daily Return
−0.2
−0.1
0.0
0.1
0.2
08
15
22
29
06
13
20
27
Oct
2008
Date
KR
PFE
XOM
WMT
DAL
CSCO
DOC
EQIX
DUK
NFLX
GE
APA
F
REGN
CMS

Table of daily returns:

Date	JPM	WFC	BAC	C	GS
2008-09-02 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2008-09-03 04:00:00+00:00	0.018466	-0.006408	0.030322	0.026164	0.013852
2008-09-03 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2008-09-04 04:00:00+00:00	-0.045329	-0.043212	-0.071602	-0.066803	-0.040034
2008-09-04 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
...	...	...	...	...	...
2008-10-28 04:00:00+00:00	0.105882	0.117743	0.121286	0.143222	0.007429
2008-10-28 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2008-10-29 04:00:00+00:00	-0.050266	-0.068195	-0.030408	-0.037286	0.043711
2008-10-29 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2008-10-30 04:00:00+00:00	0.053487	-0.008409	0.020610	0.028235	-0.067070

Date	USB	MS	KEY	PNC	COF
2008-09-02 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2008-09-03 04:00:00+00:00	0.017917	0.021066	0.008731	0.009260	0.016474
2008-09-03 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2008-09-04 04:00:00+00:00	-0.039453	-0.043396	-0.062156	-0.018484	-0.051029
2008-09-04 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
...	...	...	...	...	...
2008-10-28 04:00:00+00:00	0.069396	0.107064	0.195564	0.116152	0.159884
2008-10-28 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2008-10-29 04:00:00+00:00	-0.058079	-0.028947	0.024452	-0.040037	-0.051379
2008-10-29 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2008-10-30 04:00:00+00:00	-0.007923	0.090109	0.012346	0.026266	0.007134

Date	...	CSCO	DOC	EQIX	DUK
2008-09-02 16:00:00+00:00	...	0.000000	0.000000	0.000000	0.000000
2008-09-03 04:00:00+00:00	...	-0.018526	-0.003600	-0.009883	-0.012687
2008-09-03 16:00:00+00:00	...	0.000000	0.000000	0.000000	0.000000
2008-09-04 04:00:00+00:00	...	-0.044187	-0.040289	-0.033063	0.009930
2008-09-04 16:00:00+00:00	...	0.000000	0.000000	0.000000	0.000000
...	...	...	...	...	...
2008-10-28 04:00:00+00:00	...	0.137974	0.145642	0.101413	0.061069
2008-10-28 16:00:00+00:00	...	0.000000	0.000000	0.000000	0.000000
2008-10-29 04:00:00+00:00	...	-0.024031	-0.020020	0.047443	-0.026379
2008-10-29 16:00:00+00:00	...	0.000000	0.000000	0.000000	0.000000
2008-10-30 04:00:00+00:00	...	-0.004477	0.019407	0.050998	0.036330

Date	NFLX	GE	APA	F	REGN
2008-09-02 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2008-09-03 04:00:00+00:00	0.002270	0.001402	0.010720	0.013304	0.059281
2008-09-03 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2008-09-04 04:00:00+00:00	-0.033646	-0.030451	0.023725	-0.039387	-0.065138
2008-09-04 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
...	...	...	...	...	...
2008-10-28 04:00:00+00:00	0.146600	0.099267	0.110938	0.059113	0.083441
2008-10-28 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2008-10-29 04:00:00+00:00	0.057851	-0.014879	0.046294	0.004651	0.040597
2008-10-29 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2008-10-30 04:00:00+00:00	0.046875	0.007813	0.055340	0.055555	0.040734

```
                                CMS
Date
2008-09-02 16:00:00+00:00  0.000000
2008-09-03 04:00:00+00:00 -0.011756
2008-09-03 16:00:00+00:00  0.000000
2008-09-04 04:00:00+00:00  0.001488
2008-09-04 16:00:00+00:00  0.000000
...                             ...
2008-10-28 04:00:00+00:00  0.062047
2008-10-28 16:00:00+00:00  0.000000
2008-10-29 04:00:00+00:00 -0.009737
2008-10-29 16:00:00+00:00  0.000000
2008-10-30 04:00:00+00:00  0.036381

[82 rows x 30 columns]
```

Step 4: Correlation Matrix Computation and Visualization Using Heatmap

In [ ]:

```python
# Import necessary library for heatmap visualization
import seaborn as sns

# Step 4: Correlation Matrix Computation and Visualization

# Calculate the 30x30 correlation matrix using daily returns from Step 3
correlation_matrix = data_combined_returns.corr()

# Display the correlation matrix
print("30x30 Correlation Matrix:")
print(correlation_matrix)

# Plot a heatmap of the correlation matrix
plt.figure(figsize=(18, 14))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", linewidths=0.5)
plt.title("30x30 Correlation Matrix Heatmap for Financial and Non-Financial Institu
plt.show()
```

30x30 Correlation Matrix:

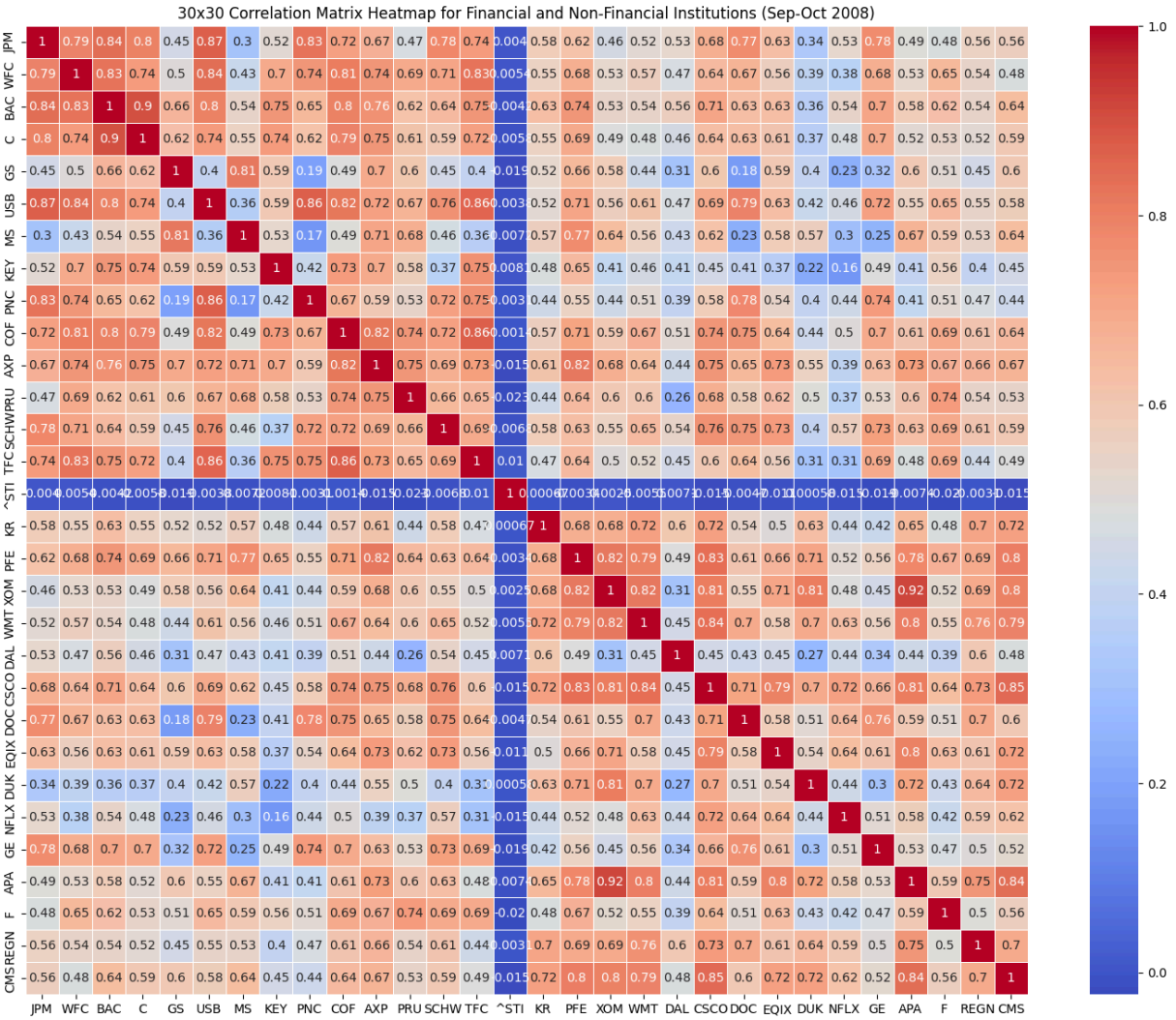
	JPM	WFC	BAC	C	GS	USB	MS	\
JPM	1.000000	0.788211	0.839790	0.799790	0.453251	0.873492	0.304182	
WFC	0.788211	1.000000	0.829078	0.735154	0.501543	0.841927	0.427621	
BAC	0.839790	0.829078	1.000000	0.904065	0.660527	0.795734	0.539347	
C	0.799790	0.735154	0.904065	1.000000	0.623111	0.735891	0.550632	
GS	0.453251	0.501543	0.660527	0.623111	1.000000	0.395515	0.814068	
USB	0.873492	0.841927	0.795734	0.735891	0.395515	1.000000	0.360371	
MS	0.304182	0.427621	0.539347	0.550632	0.814068	0.360371	1.000000	
KEY	0.516548	0.695993	0.751388	0.740148	0.587375	0.587965	0.529449	
PNC	0.831229	0.744222	0.653661	0.616543	0.194506	0.864269	0.168633	
COF	0.719514	0.805004	0.798999	0.790781	0.485907	0.821907	0.492490	
AXP	0.672407	0.742496	0.762518	0.747867	0.698119	0.716356	0.711767	
PRU	0.472492	0.694101	0.616596	0.614638	0.600593	0.666664	0.678567	
SCHW	0.783803	0.706178	0.640825	0.585221	0.448218	0.759816	0.462513	
TFC	0.738017	0.826439	0.749920	0.717660	0.395739	0.858151	0.360608	
^STI	0.003974	0.005445	-0.004177	-0.005821	-0.018922	-0.003789	-0.007162	
KR	0.584236	0.552830	0.626587	0.549393	0.524818	0.517546	0.568463	
PFE	0.619431	0.682923	0.744830	0.691125	0.655303	0.705349	0.772366	
XOM	0.460216	0.527685	0.534722	0.487672	0.581109	0.561909	0.638862	
WMT	0.519222	0.568705	0.543871	0.481207	0.438580	0.614952	0.559203	
DAL	0.526526	0.466892	0.563066	0.457169	0.314957	0.474303	0.432237	
CSCO	0.678606	0.644075	0.710125	0.636241	0.603044	0.691053	0.618699	
DOC	0.774760	0.667529	0.630355	0.629435	0.184654	0.788663	0.228728	
EQIX	0.634798	0.561325	0.628163	0.607664	0.589534	0.629216	0.578715	
DUK	0.343680	0.390810	0.364927	0.365472	0.402424	0.420730	0.571850	
NFLX	0.526594	0.375103	0.538058	0.475586	0.231867	0.459116	0.298019	
GE	0.775422	0.682423	0.702428	0.695125	0.316817	0.718216	0.249301	
APA	0.493482	0.527522	0.579876	0.515552	0.597631	0.552829	0.667085	
F	0.481702	0.653194	0.622212	0.532525	0.505758	0.645262	0.591023	
REGN	0.557818	0.541031	0.542094	0.519048	0.452372	0.546672	0.533968	
CMS	0.563532	0.478411	0.636786	0.591688	0.597629	0.575249	0.637938	

	KEY	PNC	COF	...	CSCO	DOC	EQIX	\
JPM	0.516548	0.831229	0.719514	...	0.678606	0.774760	0.634798	
WFC	0.695993	0.744222	0.805004	...	0.644075	0.667529	0.561325	
BAC	0.751388	0.653661	0.798999	...	0.710125	0.630355	0.628163	
C	0.740148	0.616543	0.790781	...	0.636241	0.629435	0.607664	
GS	0.587375	0.194506	0.485907	...	0.603044	0.184654	0.589534	
USB	0.587965	0.864269	0.821907	...	0.691053	0.788663	0.629216	
MS	0.529449	0.168633	0.492490	...	0.618699	0.228728	0.578715	
KEY	1.000000	0.423713	0.729264	...	0.454245	0.408021	0.369659	
PNC	0.423713	1.000000	0.669623	...	0.583667	0.777491	0.536694	
COF	0.729264	0.669623	1.000000	...	0.741370	0.749492	0.637569	
AXP	0.696164	0.585244	0.820284	...	0.751454	0.651347	0.728953	
PRU	0.575939	0.534397	0.741936	...	0.683213	0.575002	0.624280	
SCHW	0.374833	0.716434	0.721438	...	0.758463	0.745484	0.729591	
TFC	0.754451	0.754952	0.864739	...	0.595854	0.639511	0.562885	
^STI	0.008087	-0.003063	-0.001449	...	-0.015120	-0.004687	-0.011279	
KR	0.477620	0.444341	0.569122	...	0.716352	0.541674	0.501318	
PFE	0.645851	0.549038	0.713357	...	0.834850	0.608713	0.661203	
XOM	0.409192	0.438272	0.589547	...	0.807414	0.550379	0.707740	
WMT	0.459336	0.507324	0.674535	...	0.835499	0.697706	0.579156	
DAL	0.410850	0.392951	0.508488	...	0.451691	0.432590	0.445360	
CSCO	0.454245	0.583667	0.741370	...	1.000000	0.714862	0.792595	
DOC	0.408021	0.777491	0.749492	...	0.714862	1.000000	0.576572	

```
EQIX  0.369659  0.536694  0.637569  ...  0.792595  0.576572  1.000000
DUK   0.218101  0.402156  0.435206  ...  0.695613  0.509483  0.543072
NFLX  0.163242  0.436980  0.498384  ...  0.721956  0.641643  0.639938
GE    0.487692  0.736589  0.697189  ...  0.659871  0.763038  0.612089
APA   0.405074  0.412065  0.607985  ...  0.805073  0.586075  0.795128
F     0.556526  0.509933  0.692960  ...  0.639666  0.509785  0.626797
REGN  0.404223  0.470910  0.609743  ...  0.732961  0.699740  0.610944
CMS   0.450913  0.435176  0.641496  ...  0.845486  0.599127  0.717948
```

	DUK	NFLX	GE	APA	F	REGN	CMS
JPM	0.343680	0.526594	0.775422	0.493482	0.481702	0.557818	0.563532
WFC	0.390810	0.375103	0.682423	0.527522	0.653194	0.541031	0.478411
BAC	0.364927	0.538058	0.702428	0.579876	0.622212	0.542094	0.636786
C	0.365472	0.475586	0.695125	0.515552	0.532525	0.519048	0.591688
GS	0.402424	0.231867	0.316817	0.597631	0.505758	0.452372	0.597629
USB	0.420730	0.459116	0.718216	0.552829	0.645262	0.546672	0.575249
MS	0.571850	0.298019	0.249301	0.667085	0.591023	0.533968	0.637938
KEY	0.218101	0.163242	0.487692	0.405074	0.556526	0.404223	0.450913
PNC	0.402156	0.436980	0.736589	0.412065	0.509933	0.470910	0.435176
COF	0.435206	0.498384	0.697189	0.607985	0.692960	0.609743	0.641496
AXP	0.553064	0.388938	0.629211	0.725206	0.673373	0.663706	0.668561
PRU	0.495021	0.372881	0.530306	0.604587	0.736323	0.538093	0.532944
SCHW	0.401762	0.565584	0.727360	0.634318	0.692182	0.612028	0.588397
TFC	0.313822	0.307269	0.688925	0.483938	0.690193	0.436800	0.492353
^STI	0.000582	-0.014501	-0.019323	-0.007417	-0.019816	-0.003075	-0.015296
KR	0.625452	0.444292	0.421639	0.653072	0.478321	0.701296	0.722275
PFE	0.711589	0.516301	0.555929	0.781545	0.673231	0.689252	0.801950
XOM	0.809919	0.484656	0.450405	0.915822	0.521394	0.692944	0.795773
WMT	0.696730	0.626574	0.556923	0.802815	0.549785	0.760478	0.789510
DAL	0.265220	0.440928	0.340727	0.443706	0.393314	0.595850	0.476535
CSCO	0.695613	0.721956	0.659871	0.805073	0.639666	0.732961	0.845486
DOC	0.509483	0.641643	0.763038	0.586075	0.509785	0.699740	0.599127
EQIX	0.543072	0.639938	0.612089	0.795128	0.626797	0.610944	0.717948
DUK	1.000000	0.439007	0.302684	0.715098	0.434385	0.641697	0.720116
NFLX	0.439007	1.000000	0.507947	0.575319	0.420080	0.585788	0.618509
GE	0.302684	0.507947	1.000000	0.528249	0.467426	0.499157	0.523669
APA	0.715098	0.575319	0.528249	1.000000	0.590390	0.748921	0.839423
F	0.434385	0.420080	0.467426	0.590390	1.000000	0.495001	0.564440
REGN	0.641697	0.585788	0.499157	0.748921	0.495001	1.000000	0.696569
CMS	0.720116	0.618509	0.523669	0.839423	0.564440	0.696569	1.000000

```
[30 rows x 30 columns]
```



Step 5: Collaborative Understanding of the Upper-Confidence Bound (UCB) Algorithm

No deliverables required

Step 6: Developing and Implementing UCB Algorithm Pseudocode in Python

Develope the pseudocode a. Member A writes pseudocode that describes the UBC algorithm.

Implementing the pseudocode

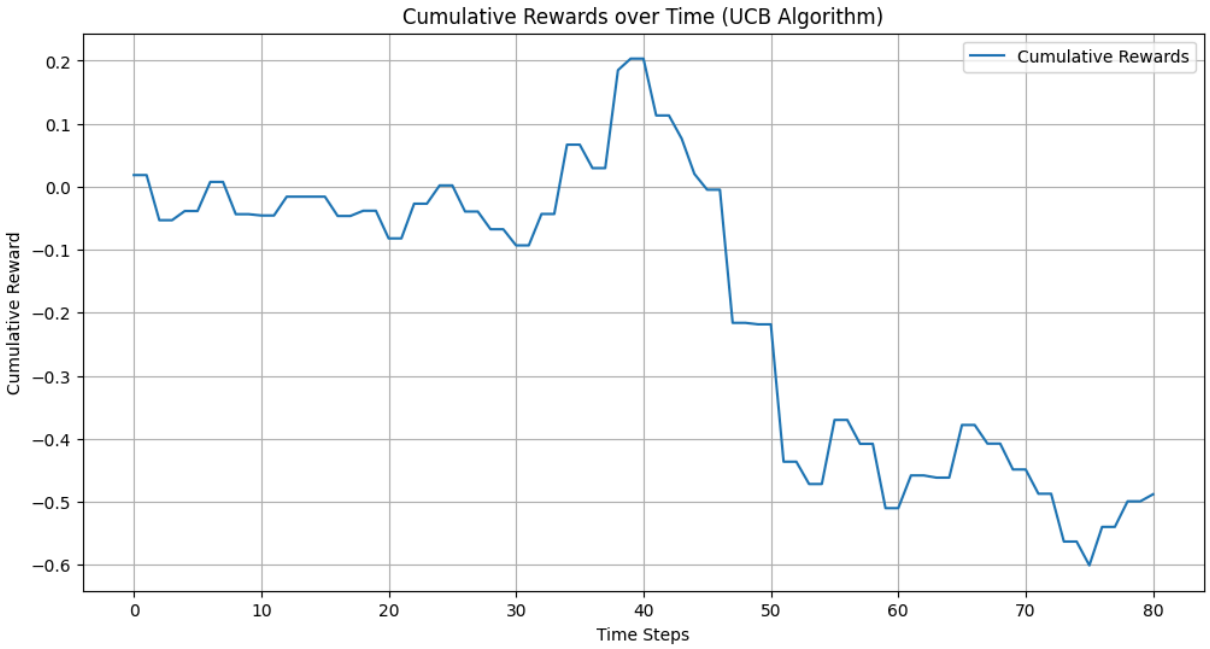
b. Using A's pseudocode, Member B implements those steps in Python. Note that member B is welcome to use Python packages.

c. Member C provides detailed comments of B's code. Member C also applies B's code to the data set.

In [ ]:
```
# Import necessary libraries (assuming previous libraries are already imported)
import numpy as np

# Step 6: Implement the UCB Algorithm using daily returns data

# Set parameters for UCB
c = 2.0  # Exploration factor to balance exploration and exploitation
num_assets = data_combined_returns.shape[1]  # Number of assets being analyzed
num_steps = len(data_combined_returns)  # Total number of time steps based on the d

# Initialize variables for UCB algorithm
Q_values = np.zeros(num_assets)  # Estimated values for each asset (mean return)
N_counts = np.zeros(num_assets)  # Count of selections for each asset to track how
reward_history = []  # Stores the rewards received at each time step for analysis
selected_assets = []  # List to track which assets are selected at each time step

# Main loop to iterate through each time step
for t in range(1, num_steps):
    UCB_scores = np.zeros(num_assets)  # Array to store UCB scores for each asset

    # Calculate UCB score for each asset
    for a in range(num_assets):
        if N_counts[a] == 0:
            UCB_scores[a] = float('inf')  # Assign infinity to ensure the asset is
        else:
            # Calculate the UCB score using the formula: Q(a) + c * sqrt(ln(t) / N(
            UCB_scores[a] = Q_values[a] + c * np.sqrt(np.log(t) / N_counts[a])

    # Select the asset with the highest UCB score
    selected_asset = np.argmax(UCB_scores)  # Identify the index of the asset with

    # Retrieve the reward (daily return) for the selected asset at the current time
    reward = data_combined_returns.iloc[t, selected_asset]  # Fetch the return for

    # Update the selection count for the chosen asset
    N_counts[selected_asset] += 1

    # Update the estimated Q-value for the chosen asset using the formula: Q(a) = Q
    Q_values[selected_asset] += (reward - Q_values[selected_asset]) / N_counts[sele

    # Record the reward received at this time step
    reward_history.append(reward)
    # Record which asset was selected
    selected_assets.append(selected_asset)

# Display the final estimated Q-values for each asset
print("Final Q-values (estimated returns) for each asset:", Q_values)
# Display the total rewards accumulated during the analysis period
print("Total rewards collected:", sum(reward_history))

# Plot the cumulative rewards over time to visualize the performance of the UCB alg
cumulative_rewards = np.cumsum(reward_history)  # Calculate cumulative sum of rewar
```

```
plt.figure(figsize=(12, 6))
plt.plot(cumulative_rewards, label="Cumulative Rewards")  # Plot the cumulative rew
plt.title("Cumulative Rewards over Time (UCB Algorithm)")  # Set the title for the
plt.xlabel("Time Steps")  # Label for the x-axis
plt.ylabel("Cumulative Reward")  # Label for the y-axis
plt.legend()  # Display the legend
plt.grid(True)  # Add grid lines for better visualization
plt.show()  # Display the plot
```

```
Final Q-values (estimated returns) for each asset: [ 0.06013803 -0.01368469 -0.08687
497  0.05183088  0.00484792 -0.00374676
  0.01539338  0.         -0.02559267 -0.04511859 -0.01875461 -0.01283954
  0.02657765 -0.00852063 -0.02803511 -0.00833337  0.02252322  0.
 -0.00965026 -0.10566299 -0.0408713  -0.02529336  0.03766915  0.01957032
  0.00966492  0.         -0.02071821 -0.10902258  0.00416648 -0.01260366]
Total rewards collected: -0.48817303615095486
```



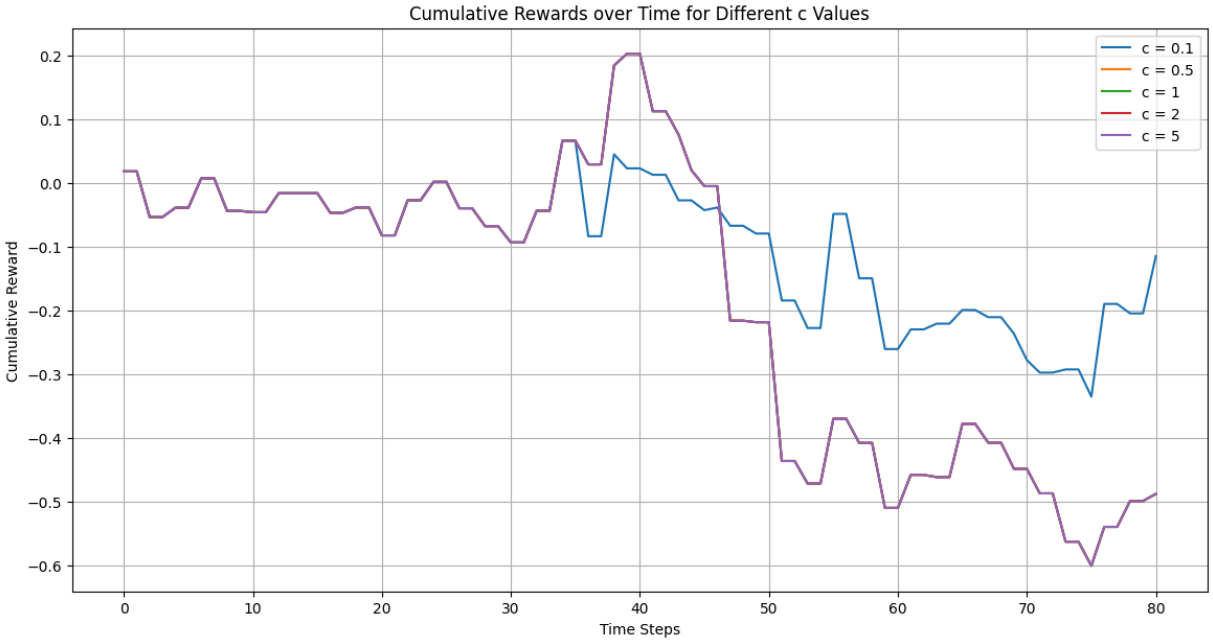
step 6 using different c

In [ ]:

```
# Import necessary libraries (assuming previous libraries are already imported)
import numpy as np
import matplotlib.pyplot as plt

# Function to implement the UCB algorithm
def ucb_algorithm(data, c):
    num_assets = data.shape[1]  # Number of assets
    Q_values = np.zeros(num_assets)  # Estimated returns for each asset
    N_counts = np.zeros(num_assets)  # Count of selections for each asset
    reward_history = []

    num_steps = len(data)
    for t in range(1, num_steps):
        UCB_scores = np.zeros(num_assets)

        for a in range(num_assets):
            if N_counts[a] == 0:
```

```
                UCB_scores[a] = float('inf')
            else:
                UCB_scores[a] = Q_values[a] + c * np.sqrt(np.log(t) / N_counts[a])

        selected_asset = np.argmax(UCB_scores)
        reward = data.iloc[t, selected_asset]
        N_counts[selected_asset] += 1
        Q_values[selected_asset] += (reward - Q_values[selected_asset]) / N_counts[
        reward_history.append(reward)

    return Q_values, reward_history

# Run the UCB algorithm for different c values
c_values = [0.1, 0.5, 1, 2, 5]
#c_values = [2]
plt.figure(figsize=(14, 7))

for c in c_values:
    Q_values, reward_history = ucb_algorithm(data_combined_returns, c)
    cumulative_rewards = np.cumsum(reward_history)

    print(f"\nResults for c = {c}")
    print("Final Q-values (estimated returns) for each asset:", Q_values)
    print("Total accumulated rewards:", sum(reward_history))

    # Plot cumulative rewards
    plt.plot(cumulative_rewards, label=f"c = {c}")

plt.title("Cumulative Rewards over Time for Different c Values")
plt.xlabel("Time Steps")
plt.ylabel("Cumulative Reward")
plt.legend()
plt.grid(True)
plt.show()
```

```
Results for c = 0.1
Final Q-values (estimated returns) for each asset: [-0.00721013  0.02665158 -0.07160
151  0.          0.00484792 -0.00504361
  0.04179421  0.00165504 -0.03120766 -0.02009396  0.03151788  0.
 -0.00714168 -0.00774126 -0.01239302 -0.01432895 -0.01530624 -0.0142431
 -0.00593656 -0.0061296  -0.01742291  0.03641057  0.00980817 -0.0525694
  0.01681804 -0.00495981 -0.02071821 -0.02163465 -0.0696332   0.        ]
Total accumulated rewards: -0.11451133636596

Results for c = 0.5
Final Q-values (estimated returns) for each asset: [ 0.06013803 -0.01368469 -0.08687
497  0.05183088  0.00484792 -0.00374676
  0.01539338  0.         -0.02559267 -0.04511859 -0.01875461 -0.01283954
  0.02657765 -0.00852063 -0.02803511 -0.00833337  0.02252322  0.
 -0.00965026 -0.10566299 -0.0408713  -0.02529336  0.03766915  0.01957032
  0.00966492  0.         -0.02071821 -0.10902258  0.00416648 -0.01260366]
Total accumulated rewards: -0.48817303615095486

Results for c = 1
Final Q-values (estimated returns) for each asset: [ 0.06013803 -0.01368469 -0.08687
497  0.05183088  0.00484792 -0.00374676
  0.01539338  0.         -0.02559267 -0.04511859 -0.01875461 -0.01283954
  0.02657765 -0.00852063 -0.02803511 -0.00833337  0.02252322  0.
 -0.00965026 -0.10566299 -0.0408713  -0.02529336  0.03766915  0.01957032
  0.00966492  0.         -0.02071821 -0.10902258  0.00416648 -0.01260366]
Total accumulated rewards: -0.48817303615095486

Results for c = 2
Final Q-values (estimated returns) for each asset: [ 0.06013803 -0.01368469 -0.08687
497  0.05183088  0.00484792 -0.00374676
  0.01539338  0.         -0.02559267 -0.04511859 -0.01875461 -0.01283954
  0.02657765 -0.00852063 -0.02803511 -0.00833337  0.02252322  0.
 -0.00965026 -0.10566299 -0.0408713  -0.02529336  0.03766915  0.01957032
  0.00966492  0.         -0.02071821 -0.10902258  0.00416648 -0.01260366]
Total accumulated rewards: -0.48817303615095486

Results for c = 5
Final Q-values (estimated returns) for each asset: [ 0.06013803 -0.01368469 -0.08687
497  0.05183088  0.00484792 -0.00374676
  0.01539338  0.         -0.02559267 -0.04511859 -0.01875461 -0.01283954
  0.02657765 -0.00852063 -0.02803511 -0.00833337  0.02252322  0.
 -0.00965026 -0.10566299 -0.0408713  -0.02529336  0.03766915  0.01957032
  0.00966492  0.         -0.02071821 -0.10902258  0.00416648 -0.01260366]
Total accumulated rewards: -0.48817303615095486
```



Step 7: Collaborative Discussion on Epsilon-Greedy Algorithm for Portfolio Selection

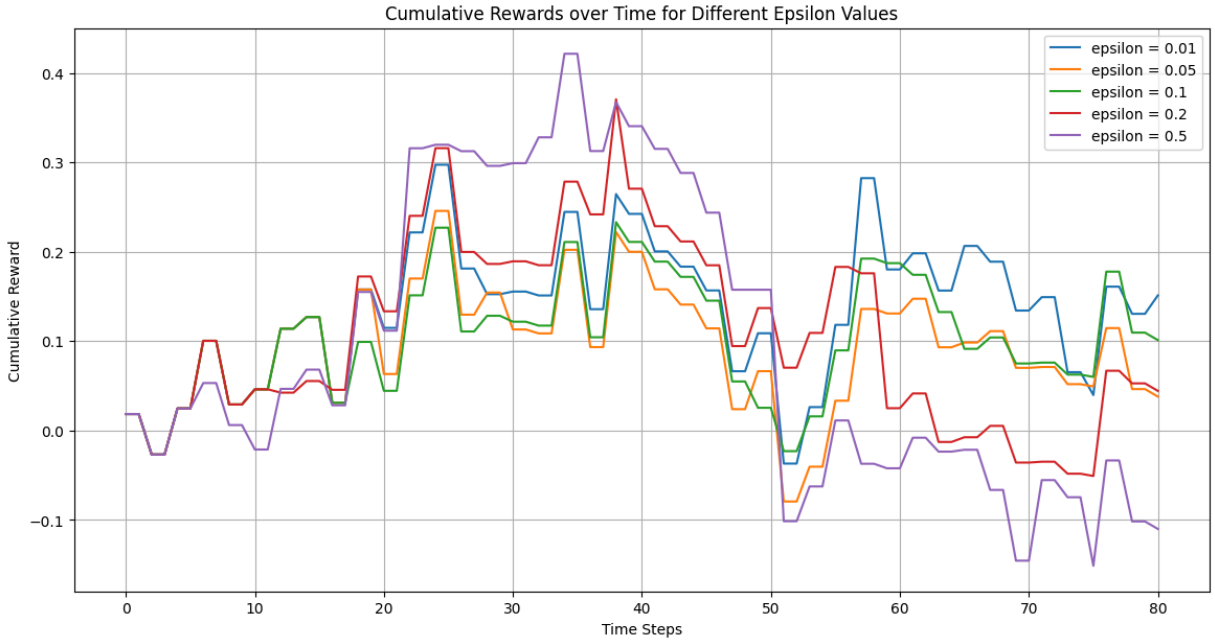
No deliverables required

Step 8: Pseudocode and Python Implementation of the Epsilon-Greedy Algorithm

In [ ]:
```python
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset prepared in Step 3
# Assuming 'data_combined_returns' holds the daily returns for each asset

def epsilon_greedy_algorithm(data, epsilon):
    # Initialize parameters
    num_assets = data.shape[1]  # Number of assets
    Q_values = np.zeros(num_assets)  # Estimated returns for each asset
    N_counts = np.zeros(num_assets)  # Count of selections for each asset
    reward_history = []  # List to track the reward history

    # Main loop over time steps
    num_steps = len(data)
    for t in range(1, num_steps):
```

```
        # Generate random number for exploration decision
        r = np.random.rand()

        if r < epsilon:
            # Exploration: select a random asset
            selected_asset = np.random.randint(0, num_assets)
        else:
            # Exploitation: select the asset with the highest Q_value
            selected_asset = np.argmax(Q_values)

        # Retrieve the reward for the selected asset at time t
        reward = data.iloc[t, selected_asset]

        # Update the selection count for the chosen asset
        N_counts[selected_asset] += 1

        # Update Q_value for the chosen asset
        Q_values[selected_asset] += (1 / N_counts[selected_asset]) * (reward - Q_va

        # Append the reward to the reward history
        reward_history.append(reward)

    # Return results
    return Q_values, reward_history

# Run the epsilon-greedy algorithm for different epsilon values
epsilon_values = [0.01,0.05,0.1,0.2,0.5]
#epsilon_values = [0.2]
results = {}
plt.figure(figsize=(14, 7))

for epsilon in epsilon_values:
    Q_values, reward_history = epsilon_greedy_algorithm(data_combined_returns, epsi
    cumulative_rewards = np.cumsum(reward_history)
    results[epsilon] = (Q_values, cumulative_rewards)

    # Print the final Q-values and total rewards
    print(f"\nResults for epsilon = {epsilon}")
    print("Final Q-values (estimated returns) for each asset:", Q_values)
    print("Total accumulated rewards:", sum(reward_history))

    # Plot the cumulative rewards on the same graph
    plt.plot(cumulative_rewards, label=f"epsilon = {epsilon}")

plt.title("Cumulative Rewards over Time for Different Epsilon Values")
plt.xlabel("Time Steps")
plt.ylabel("Cumulative Reward")
plt.legend()
plt.grid(True)
plt.show()
```

```
Results for epsilon = 0.01
Final Q-values (estimated returns) for each asset: [-0.00895433 -0.00021935  0.00648
678  0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.        ]
Total accumulated rewards: 0.15116382648589444

Results for epsilon = 0.05
Final Q-values (estimated returns) for each asset: [-0.00895433  0.00190986  0.
0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.         -0.0055227   0.          0.          0.         -0.01917842
  0.         -0.01464652  0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.        ]
Total accumulated rewards: 0.03796797217412229

Results for epsilon = 0.1
Final Q-values (estimated returns) for each asset: [-0.00671574  0.00725057 -0.04164
926  0.         -0.02200733  0.00335486
  0.          0.          0.          0.          0.          0.
 -0.02881235  0.          0.          0.         -0.02946145  0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.         -0.01806542 -0.02909088  0.        ]
Total accumulated rewards: 0.10112401054198128

Results for epsilon = 0.2
Final Q-values (estimated returns) for each asset: [-0.02338237  0.00854322  0.
-0.00374708  0.          0.
  0.          0.          0.         -0.08214725  0.         -0.09999986
 -0.01412403  0.          0.         -0.03778313  0.          0.
  0.         -0.00492023  0.          0.          0.          0.
  0.         -0.00714346  0.          0.          0.          0.        ]
Total accumulated rewards: 0.044348258959429177

Results for epsilon = 0.5
Final Q-values (estimated returns) for each asset: [-0.00895433  0.01807772  0.
0.         -0.02691154 -0.08620661
 -0.08630954  0.         -0.02731626  0.          0.01421326  0.
  0.          0.01814571  0.          0.         -0.02351111  0.01773896
 -0.00778122  0.         -0.01644338  0.00801427 -0.02512474  0.
  0.00357614  0.         -0.01326505  0.          0.00130549 -0.04240399]
Total accumulated rewards: -0.11007745284122727
```

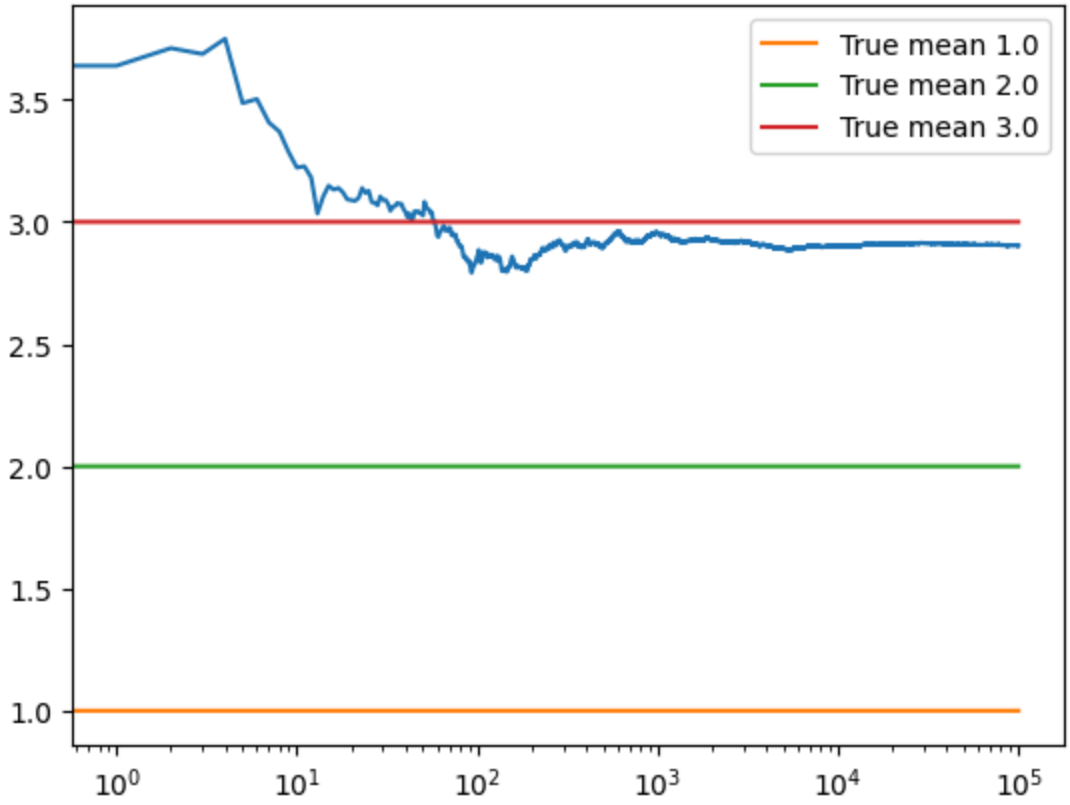


Epsilon

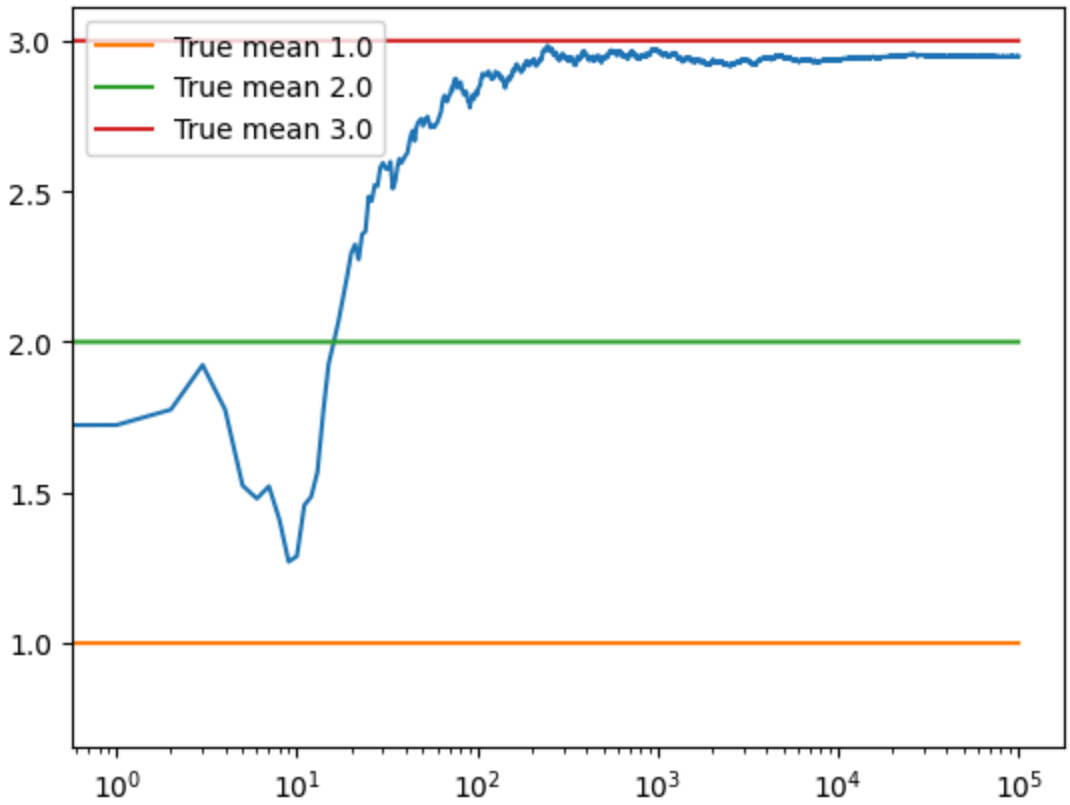
In [ ]:
```python
# Import required libraries
import numpy as np
import matplotlib.pyplot as plt

# Define Action class
class Actions:
    def __init__(self, m):
        self.m = m
        self.mean = 0
        self.N = 0

    # Choose a random action
    def choose(self):
        return np.random.randn() + self.m

    # Update the action-value estimate
    def update(self, x):
        self.N += 1
        self.mean = (1 - 1.0 / self.N) * self.mean + (1.0 / self.N) * x

# Run experiment function
def run_experiment(m1, m2, m3, eps, N):
    actions = [Actions(m1), Actions(m2), Actions(m3)]
    data = np.empty(N)

    for i in range(N):
        # epsilon-greedy choice
        p = np.random.random()
        if p < eps:
            j = np.random.choice(3)
        else:
            j = np.argmax([a.mean for a in actions])
        x = actions[j].choose()
```

```
        actions[j].update(x)

        # For plotting
        data[i] = x

    cumulative_average = np.cumsum(data) / (np.arange(N) + 1)

    # Plot moving average CTR
    plt.plot(cumulative_average)
    plt.plot(np.ones(N) * m1, label=f"True mean {m1}")
    plt.plot(np.ones(N) * m2, label=f"True mean {m2}")
    plt.plot(np.ones(N) * m3, label=f"True mean {m3}")
    plt.xscale('log')
    plt.legend()
    plt.show()

    # Print final estimated means
    for idx, a in enumerate(actions):
        print(f"Final estimated mean for action {idx + 1}: {a.mean:.4f}")

    return cumulative_average

# Main execution
if __name__ == '__main__':
    print("Running experiment with epsilon = 0.1")
    c_1 = run_experiment(1.0, 2.0, 3.0, 0.1, 100000)

    print("\nRunning experiment with epsilon = 0.05")
    c_05 = run_experiment(1.0, 2.0, 3.0, 0.05, 100000)

    print("\nRunning experiment with epsilon = 0.01")
    c_01 = run_experiment(1.0, 2.0, 3.0, 0.01, 100000)
```

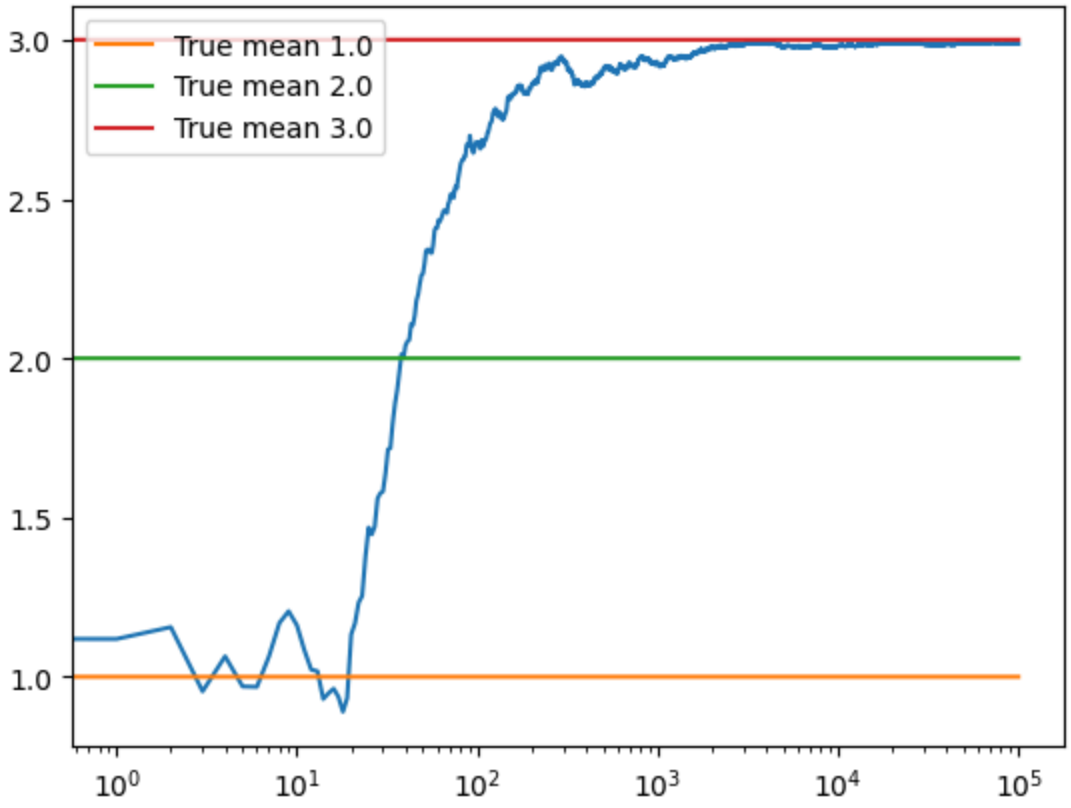
```
Running experiment with epsilon = 0.1
```




```
Final estimated mean for action 1: 1.0131
Final estimated mean for action 2: 1.9839
Final estimated mean for action 3: 3.0035

Running experiment with epsilon = 0.05
```

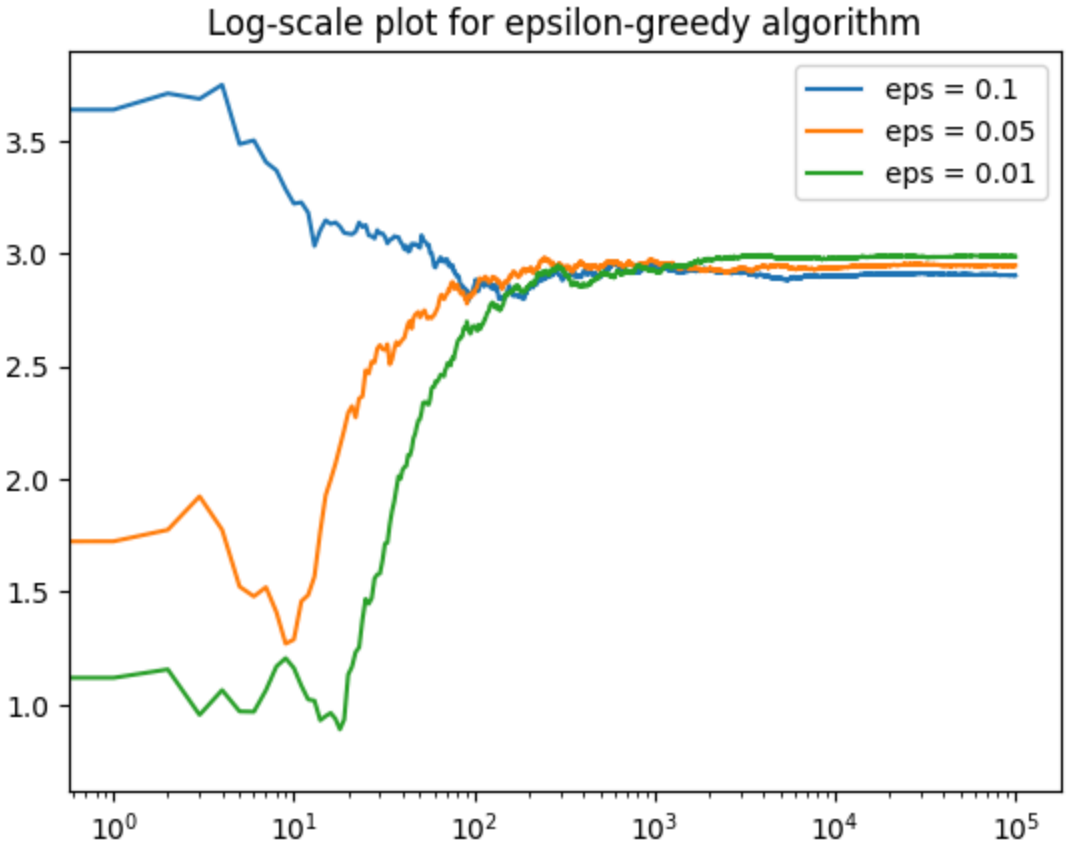


```
Final estimated mean for action 1: 0.9829
Final estimated mean for action 2: 1.9914
Final estimated mean for action 3: 2.9990

Running experiment with epsilon = 0.01
```

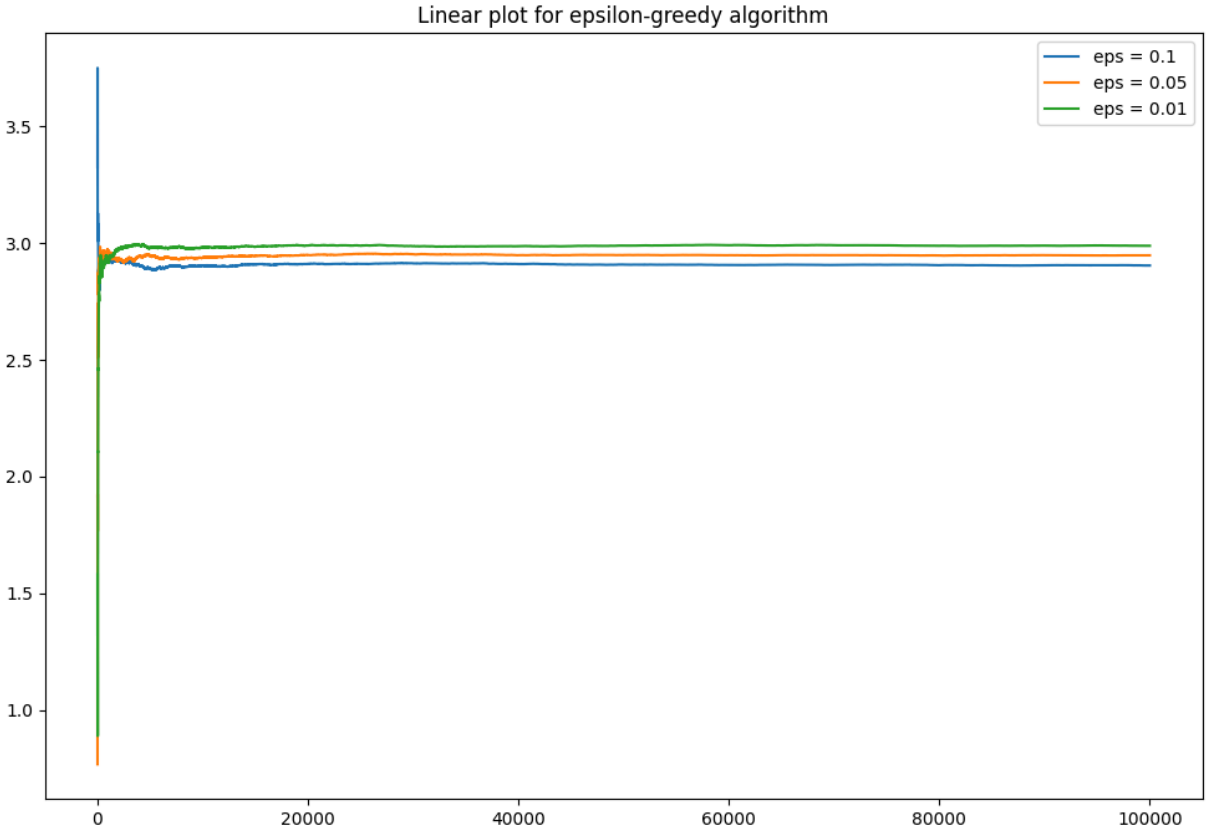



```
Final estimated mean for action 1: 1.0054
Final estimated mean for action 2: 2.0810
Final estimated mean for action 3: 2.9982
```

In [ ]:
```python
# log scale plot
plt.plot(c_1, label ='eps = 0.1')
plt.plot(c_05, label ='eps = 0.05')
plt.plot(c_01, label ='eps = 0.01')
plt.legend()
plt.title("Log-scale plot for epsilon-greedy algorithm")
plt.xscale('log')
plt.show()
```



In [ ]:
```
# Linear plot
plt.figure(figsize = (12, 8))
plt.plot(c_1, label ='eps = 0.1')
plt.plot(c_05, label ='eps = 0.05')
plt.plot(c_01, label ='eps = 0.01')
plt.legend()
plt.title("Linear plot for epsilon-greedy algorithm")
plt.show()
```

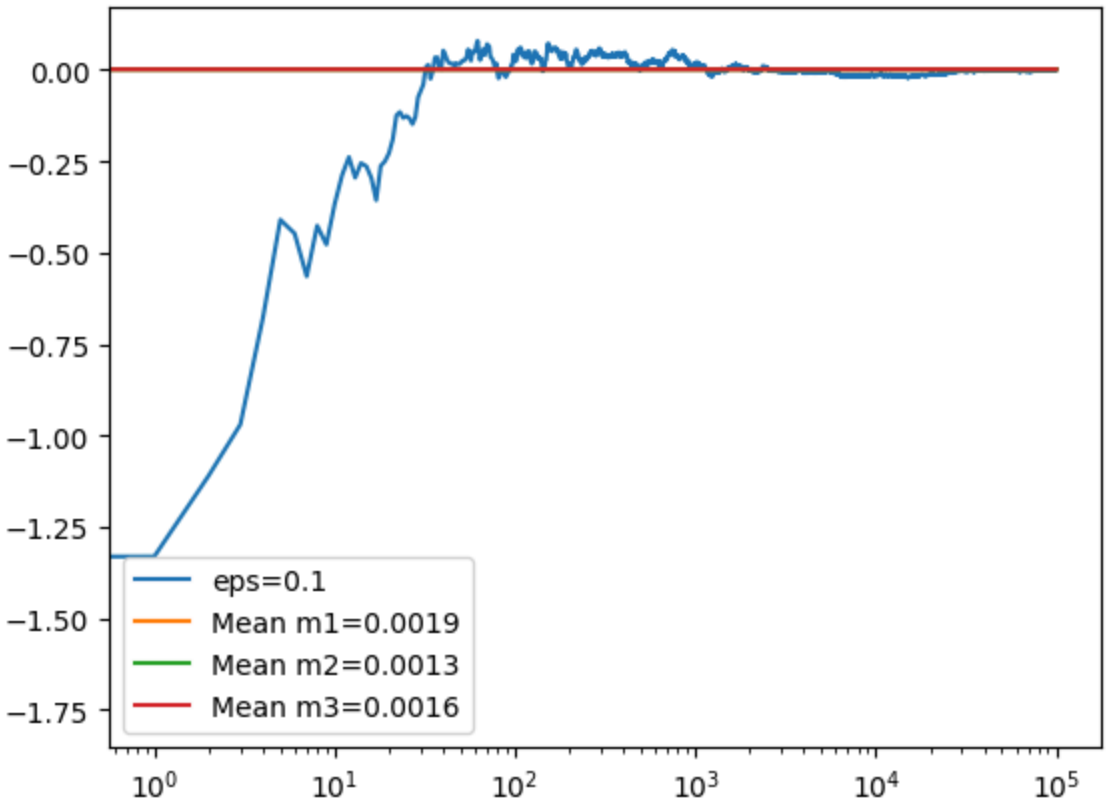


In [ ]:

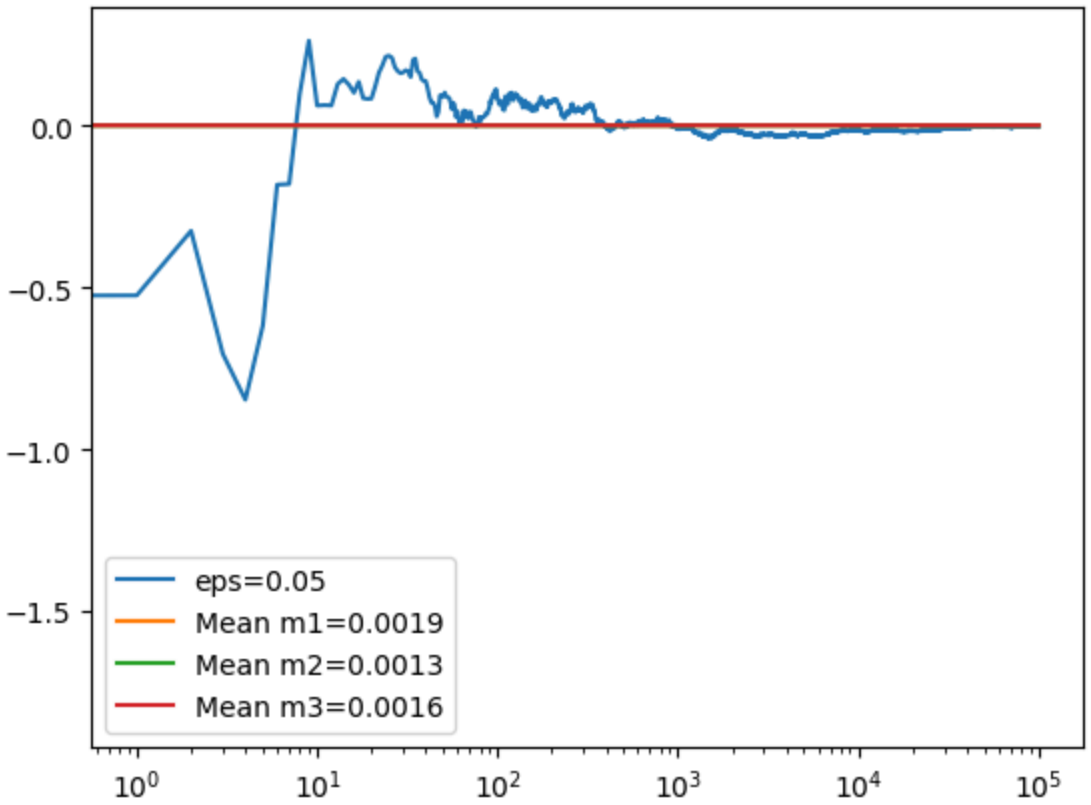
```
import numpy as np
import matplotlib.pyplot as plt
import yfinance as yf

# Load AMZN stock data
data = yf.download('AMZN', start='2009-01-01', end='2024-01-01')
data['Daily Return'] = data['Adj Close'].pct_change()

# Calculate means for different market conditions (these are hypothetical and can b
m1 = data['Daily Return'][:1000].mean()  # Mean of first 1000 days
m2 = data['Daily Return'][1000:2000].mean()  # Mean of next 1000 days
m3 = data['Daily Return'][2000:3000].mean()  # Mean of the following 1000 days

# Define Action class
class Actions:
    def __init__(self, m):
        self.m = m  # Mean based on market condition
        self.mean = 0
        self.N = 0

    def choose(self):
        return np.random.randn() + self.m

    def update(self, x):
        self.N += 1
        self.mean = (1 - 1.0 / self.N)*self.mean + 1.0 / self.N * x

# Experiment function using real data-driven means
def run_experiment(m1, m2, m3, eps, N):
    actions = [Actions(m1), Actions(m2), Actions(m3)]
```

```
    data = np.empty(N)

    for i in range(N):
        # epsilon greedy
        p = np.random.random()
        if p < eps:
            j = np.random.choice(3)
        else:
            j = np.argmax([a.mean for a in actions])
        x = actions[j].choose()
        actions[j].update(x)

        # For plotting
        data[i] = x
    cumulative_average = np.cumsum(data) / (np.arange(N) + 1)

    # Plot cumulative average return
    plt.plot(cumulative_average, label=f'eps={eps}')
    plt.plot(np.ones(N) * m1, label=f'Mean m1={m1:.4f}')
    plt.plot(np.ones(N) * m2, label=f'Mean m2={m2:.4f}')
    plt.plot(np.ones(N) * m3, label=f'Mean m3={m3:.4f}')
    plt.xscale('log')
    plt.legend()
    plt.show()

    for a in actions:
        print(f'Final estimated mean: {a.mean}')

    return cumulative_average

# Run experiments with different epsilon values
if __name__ == '__main__':
    c_1 = run_experiment(m1, m2, m3, 0.1, 100000)
    c_05 = run_experiment(m1, m2, m3, 0.05, 100000)
    c_01 = run_experiment(m1, m2, m3, 0.01, 100000)
```

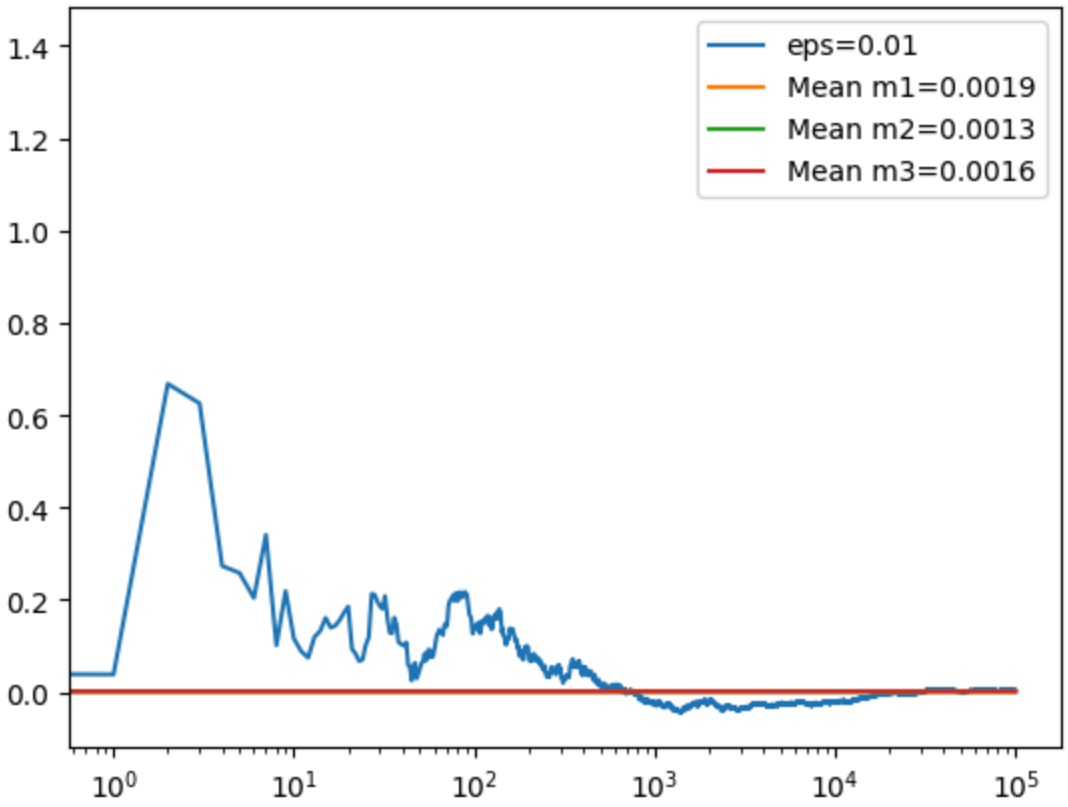
```
[**********100%**********]  1 of 1 completed
```




```
Final estimated mean: 0.004041565144553345
Final estimated mean: -0.010640158629187485
Final estimated mean: -0.008834692248067854
```

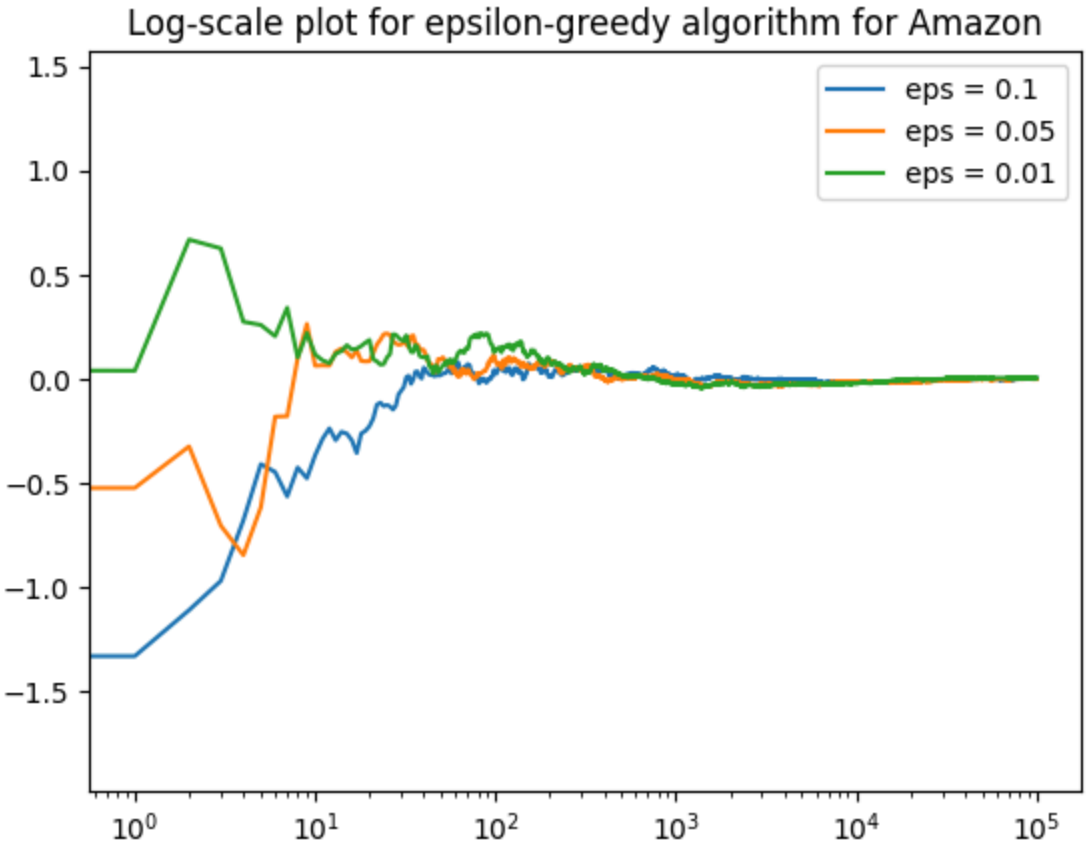



```
Final estimated mean: -0.004712839972321432
Final estimated mean: -0.005269651920907604
Final estimated mean: 0.0013693674404622314
```



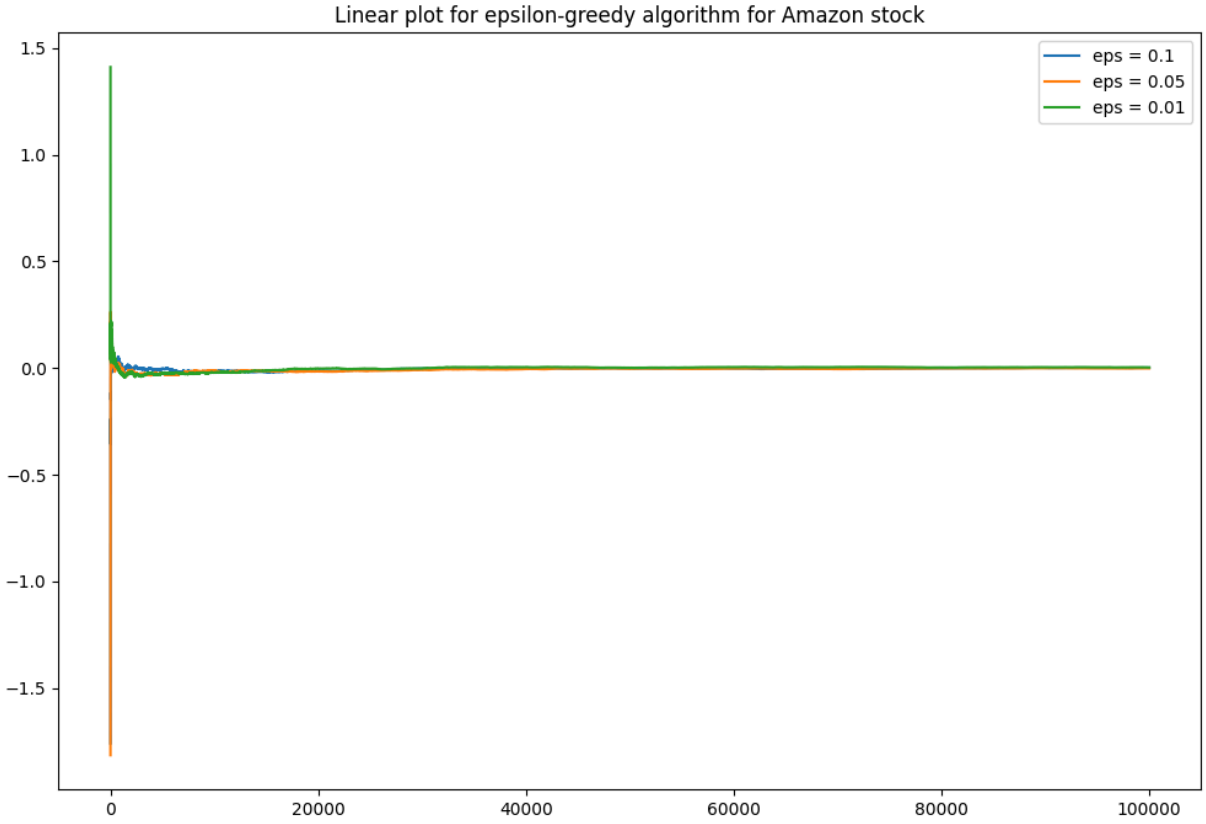

```
Final estimated mean: -0.006751526833698428
Final estimated mean: -0.06307215847761821
Final estimated mean: 0.0034566972360582774
```

In [ ]:
```
# log scale plot
plt.plot(c_1, label ='eps = 0.1')
plt.plot(c_05, label ='eps = 0.05')
plt.plot(c_01, label ='eps = 0.01')
plt.legend()
plt.title("Log-scale plot for epsilon-greedy algorithm for Amazon")
plt.xscale('log')
plt.show()
```



In [ ]:

In [ ]:
```
# linear plot
plt.figure(figsize = (12, 8))
plt.plot(c_1, label ='eps = 0.1')
plt.plot(c_05, label ='eps = 0.05')
plt.plot(c_01, label ='eps = 0.01')
plt.legend()
plt.title("Linear plot for epsilon-greedy algorithm for Amazon stock")
plt.show()
```



Step 9: Comparative Analysis of Results: UCB vs. Epsilon-Greedy and Huo's Paper

Results from Huo and Fu

From their study (Huo and Fu), the Monte Carlo simulation was used to evaluate the performance of a sequential portfolio selection algorithm that incorporated the Upper Confidence Bound (UCB) algorithm, specifically the UCB1 policy, as an exploration-focused component. The UCB1 portfolio was compared against four benchmarks: a risk-aware portfolio, an ε-greedy portfolio, an equally weighted portfolio, and a combined portfolio that balanced the UCB1 and risk-aware portfolios. The results showed that the UCB1 portfolio consistently achieved the highest cumulative wealth, outperforming all other benchmarks. Regardless, there was high variability in the performance of the UCB1 portfolio. In volatile markets, the UCB1 policy seemed to have taken a longer time to reach optimality.

Step 10: Updating Data Series for Financial and Non-Financial Companies

In [ ]:
```
# Import necessary libraries
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Step 10: Define the updated date range for March to April 2020
start_date_update = '2020-03-01'
end_date_update = '2020-04-30'

# Define the lists of financial and non-financial institutions
financial_institutions = ["JPM", "WFC", "BAC", "C", "GS", "USB", "MS", "KEY", "PNC"
non_financial_institutions = ["KR", "PFE", "XOM", "WMT", "DAL", "CSCO", "DOC", "EQI

# Function to get historical data from Yahoo Finance
def get_historical_data(symbols, start, end):
    data_dict = {}
    for symbol in symbols:
        try:
            stock_data = yf.Ticker(symbol).history(start=start, end=end)
            if not stock_data.empty:
                data_dict[symbol] = stock_data['Close']  # Collect only the closing
                print(f"Data for {symbol} retrieved successfully.")
            else:
                print(f"No data found for {symbol}.")
        except Exception as e:
            print(f"Error retrieving data for {symbol}: {e}")
    return pd.DataFrame(data_dict)

# Collect updated historical data for financial institutions (March-April 2020)
financial_data_updated = get_historical_data(financial_institutions, start_date_upd

# Collect updated historical data for non-financial institutions (March-April 2020)
non_financial_data_updated = get_historical_data(non_financial_institutions, start_

# Merge updated data into a unified time series
data_combined_updated = pd.concat([financial_data_updated, non_financial_data_updat

# Calculate daily returns for each company in the updated dataset
data_combined_returns_updated = data_combined_updated.pct_change().dropna()

# Save the updated data to CSV files for reporting purposes
data_combined_updated.to_csv("updated_historical_prices_combined.csv")
data_combined_returns_updated.to_csv("updated_daily_returns_combined.csv")

# Display a sample of the updated data
print("Sample of updated historical prices:")
print(data_combined_updated.head())

print("\nSample of updated daily returns:")
print(data_combined_returns_updated.head())

# Plot historical prices for financial institutions (March-April 2020)
financial_data_updated.plot(figsize=(16, 9), title="Historical Prices of Financial
plt.xlabel("Date")
plt.ylabel("Price (USD)")
plt.legend(loc='upper right')
plt.show()

# Plot daily returns for financial institutions (March-April 2020)
financial_data_returns_updated = financial_data_updated.pct_change().dropna()
```

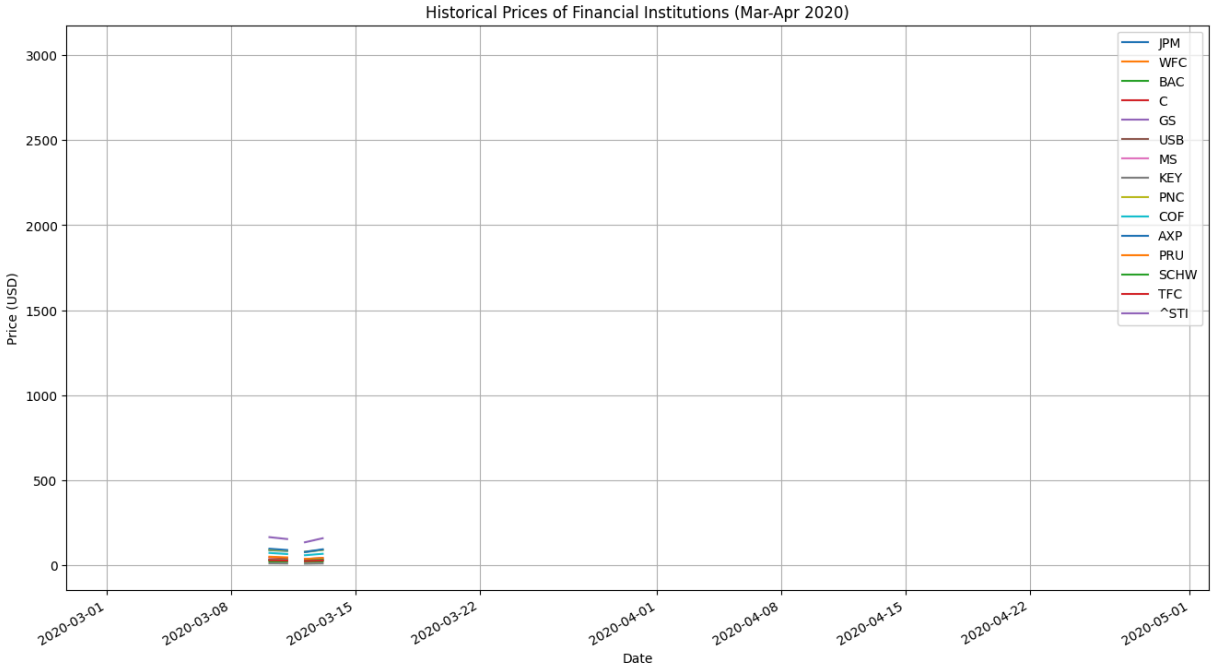
```
financial_data_returns_updated.plot(figsize=(16, 9), title="Daily Returns of Financ
plt.xlabel("Date")
plt.ylabel("Daily Return")
plt.legend(loc='upper right')
plt.show()

# Plot historical prices for non-financial institutions (March-April 2020)
non_financial_data_updated.plot(figsize=(16, 9), title="Historical Prices of Non-Fi
plt.xlabel("Date")
plt.ylabel("Price (USD)")
plt.legend(loc='upper right')
plt.show()

# Plot daily returns for non-financial institutions (March-April 2020)
non_financial_data_returns_updated = non_financial_data_updated.pct_change().dropna
non_financial_data_returns_updated.plot(figsize=(16, 9), title="Daily Returns of No
plt.xlabel("Date")
plt.ylabel("Daily Return")
plt.legend(loc='upper right')
plt.show()

# Display daily returns in a table format for reporting
print("\nTable of updated daily returns:")
print(data_combined_returns_updated)
```

```
Data for JPM retrieved successfully.
Data for WFC retrieved successfully.
Data for BAC retrieved successfully.
Data for C retrieved successfully.
Data for GS retrieved successfully.
Data for USB retrieved successfully.
Data for MS retrieved successfully.
Data for KEY retrieved successfully.
Data for PNC retrieved successfully.
Data for COF retrieved successfully.
Data for AXP retrieved successfully.
Data for PRU retrieved successfully.
Data for SCHW retrieved successfully.
Data for TFC retrieved successfully.
Data for ^STI retrieved successfully.
Data for KR retrieved successfully.
Data for PFE retrieved successfully.
Data for XOM retrieved successfully.
Data for WMT retrieved successfully.
Data for DAL retrieved successfully.
Data for CSCO retrieved successfully.
Data for DOC retrieved successfully.
Data for EQIX retrieved successfully.
Data for DUK retrieved successfully.
Data for NFLX retrieved successfully.
Data for GE retrieved successfully.
Data for APA retrieved successfully.
Data for F retrieved successfully.
Data for REGN retrieved successfully.
Data for CMS retrieved successfully.
Sample of updated historical prices:
                                  JPM        WFC        BAC          C  \
Date
2020-03-01 16:00:00+00:00         NaN        NaN        NaN        NaN
2020-03-02 05:00:00+00:00  105.843155  37.604839  26.080284  56.222092
2020-03-02 16:00:00+00:00         NaN        NaN        NaN        NaN
2020-03-03 05:00:00+00:00  101.871437  36.065430  24.641735  54.109291
2020-03-03 16:00:00+00:00         NaN        NaN        NaN        NaN

                                   GS        USB         MS        KEY  \
Date
2020-03-01 16:00:00+00:00         NaN        NaN        NaN        NaN
2020-03-02 05:00:00+00:00  187.398666  39.667667  39.804768  13.560563
2020-03-02 16:00:00+00:00         NaN        NaN        NaN        NaN
2020-03-03 05:00:00+00:00  181.995071  37.970909  38.023491  12.953253
2020-03-03 16:00:00+00:00         NaN        NaN        NaN        NaN

                                  PNC        COF  ...       CSCO        DOC  \
Date                                              ...
2020-03-01 16:00:00+00:00         NaN        NaN  ...        NaN        NaN
2020-03-02 05:00:00+00:00  111.073219  84.159622  ...  35.759884  25.898697
2020-03-02 16:00:00+00:00         NaN        NaN  ...        NaN        NaN
2020-03-03 05:00:00+00:00  105.085777  79.513252  ...  34.778381  26.214243
2020-03-03 16:00:00+00:00         NaN        NaN  ...        NaN        NaN

                                 EQIX        DUK        NFLX         GE  \
```

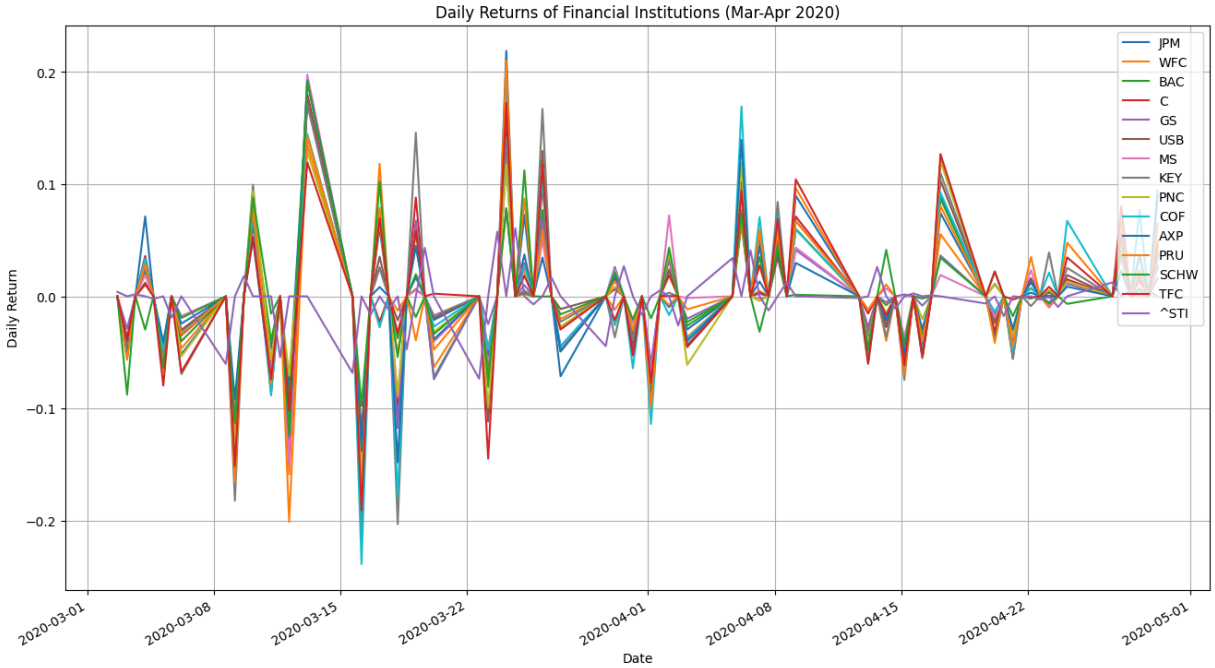
```
Date
2020-03-01 16:00:00+00:00         NaN        NaN         NaN        NaN
2020-03-02 05:00:00+00:00  571.367798  80.419876  381.049988  54.740047
2020-03-02 16:00:00+00:00         NaN        NaN         NaN        NaN
2020-03-03 05:00:00+00:00  567.933228  79.538063  368.769989  53.128613
2020-03-03 16:00:00+00:00         NaN        NaN         NaN        NaN

                                  APA         F        REGN        CMS
Date
2020-03-01 16:00:00+00:00         NaN       NaN         NaN        NaN
2020-03-02 05:00:00+00:00  23.359791  5.947534  464.750000  55.985958
2020-03-02 16:00:00+00:00         NaN       NaN         NaN        NaN
2020-03-03 05:00:00+00:00  22.874657  5.757544  461.549988  56.143070
2020-03-03 16:00:00+00:00         NaN       NaN         NaN        NaN

[5 rows x 30 columns]

Sample of updated daily returns:
                                 JPM       WFC       BAC         C        GS  \
Date
2020-03-02 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000
2020-03-03 05:00:00+00:00 -0.037525 -0.040936 -0.055158 -0.037580 -0.028835
2020-03-03 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000
2020-03-04 05:00:00+00:00  0.024709  0.021465  0.023063  0.035972  0.026102
2020-03-04 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000

                                 USB        MS       KEY       PNC       COF  \
Date
2020-03-02 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000
2020-03-03 05:00:00+00:00 -0.042774 -0.044750 -0.044785 -0.053905 -0.055209
2020-03-03 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000
2020-03-04 05:00:00+00:00  0.012148  0.018919  0.029611  0.031158  0.033673
2020-03-04 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000

                           ...      CSCO       DOC      EQIX       DUK  \
Date                       ...
2020-03-02 16:00:00+00:00  ...  0.000000  0.000000  0.000000  0.000000
2020-03-03 05:00:00+00:00  ... -0.027447  0.012184 -0.006011 -0.010965
2020-03-03 16:00:00+00:00  ...  0.000000  0.000000  0.000000  0.000000
2020-03-04 05:00:00+00:00  ...  0.033716  0.040325  0.049304  0.063173
2020-03-04 16:00:00+00:00  ...  0.000000  0.000000  0.000000  0.000000

                                NFLX        GE       APA         F      REGN  \
Date
2020-03-02 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000
2020-03-03 05:00:00+00:00 -0.032227 -0.029438 -0.020768 -0.031944 -0.006885
2020-03-03 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000
2020-03-04 05:00:00+00:00  0.040730  0.006434  0.005602  0.015782  0.069180
2020-03-04 16:00:00+00:00  0.000000  0.000000  0.000000  0.000000  0.000000

                                 CMS
Date
2020-03-02 16:00:00+00:00  0.000000
2020-03-03 05:00:00+00:00  0.002806
2020-03-03 16:00:00+00:00  0.000000
2020-03-04 05:00:00+00:00  0.062655
```

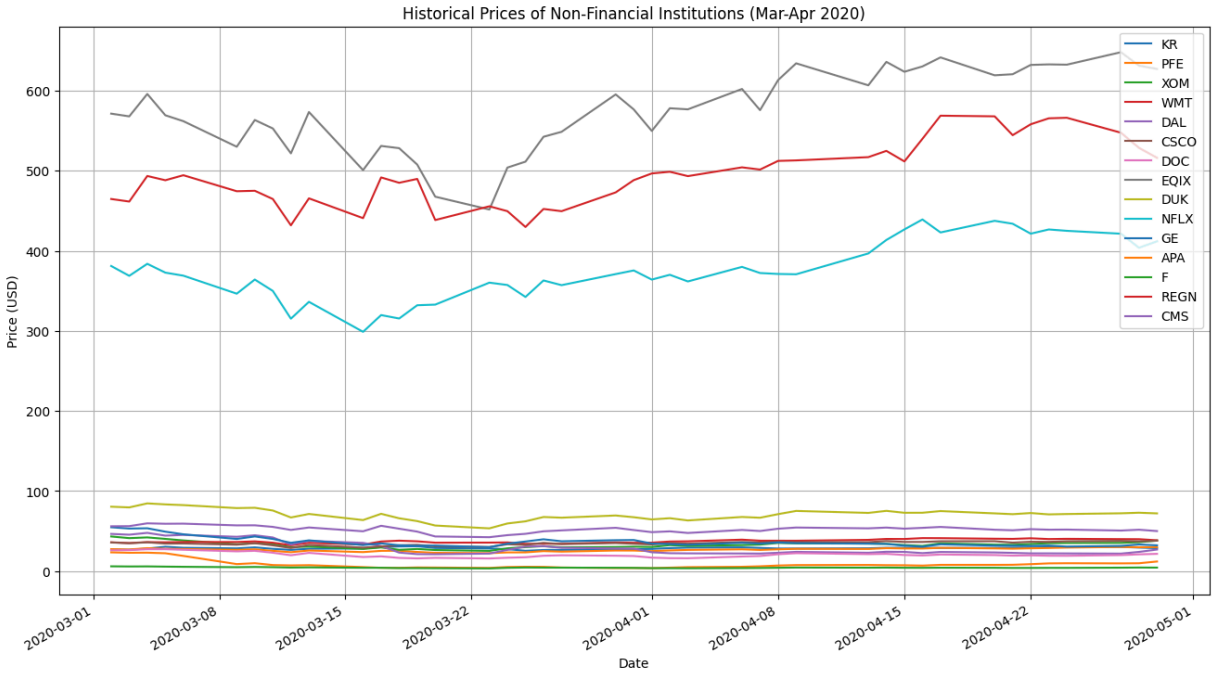
```
2020-03-04 16:00:00+00:00  0.000000

[5 rows x 30 columns]
```

```
<ipython-input-12-0099c9bccdb2>:39: FutureWarning: The default fill_method='pad' in
DataFrame.pct_change is deprecated and will be removed in a future version. Either f
ill in any non-leading NA values prior to calling pct_change or specify 'fill_method
=None' to not fill NA values.
  data_combined_returns_updated = data_combined_updated.pct_change().dropna()
```




```
<ipython-input-12-0099c9bccdb2>:60: FutureWarning: The default fill_method='pad' in
DataFrame.pct_change is deprecated and will be removed in a future version. Either f
ill in any non-leading NA values prior to calling pct_change or specify 'fill_method
=None' to not fill NA values.
  financial_data_returns_updated = financial_data_updated.pct_change().dropna()
```




Historical Prices of Non-Financial Institutions (Mar-Apr 2020)
Price (USD)
Date
KR
PFE
XOM
WMT
DAL
CSCO
DOC
EQIX
DUK
NFLX
GE
APA
F
REGN
CMS

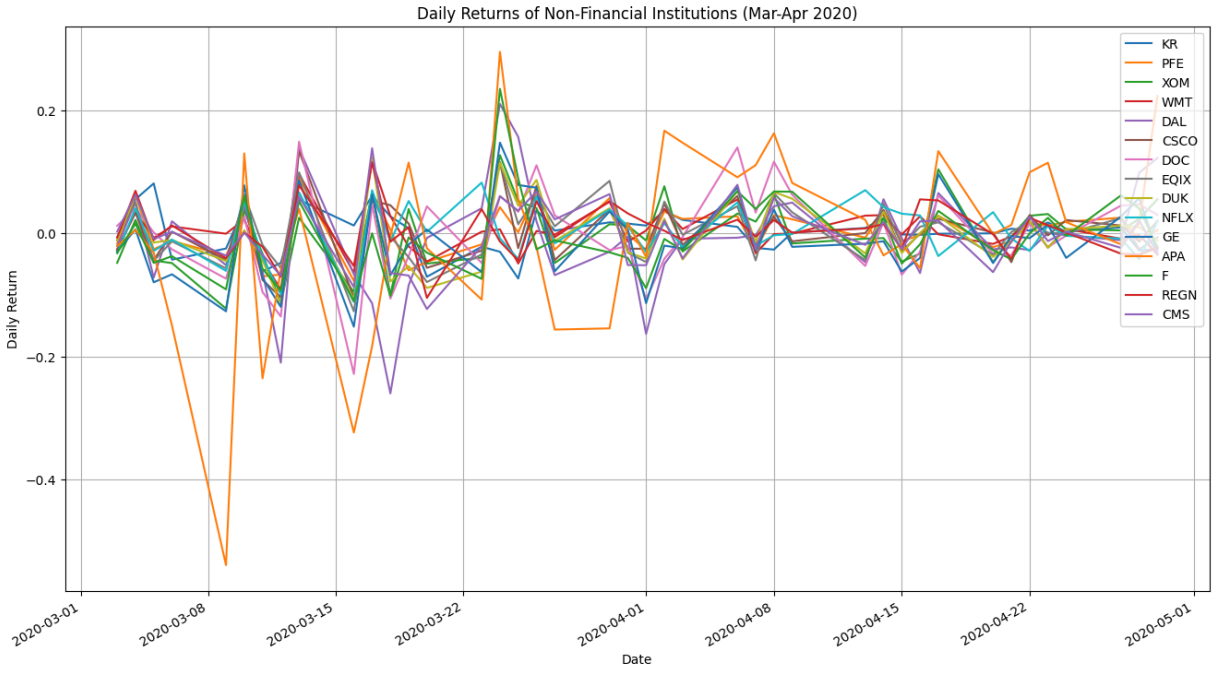

Daily Returns of Non-Financial Institutions (Mar-Apr 2020)
Daily Return
Date
KR
PFE
XOM
WMT
DAL
CSCO
DOC
EQIX
DUK
NFLX
GE
APA
F
REGN
CMS

Table of updated daily returns:

Date	JPM	WFC	BAC	C	GS
2020-03-02 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2020-03-03 05:00:00+00:00	-0.037525	-0.040936	-0.055158	-0.037580	-0.028835
2020-03-03 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2020-03-04 05:00:00+00:00	0.024709	0.021465	0.023063	0.035972	0.026102
2020-03-04 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
...	...	...	...	...	...
2020-04-27 04:00:00+00:00	0.043104	0.055349	0.058160	0.080278	0.036949
2020-04-27 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2020-04-28 04:00:00+00:00	0.007081	0.016191	0.017895	0.013961	0.018906
2020-04-28 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2020-04-29 04:00:00+00:00	0.026970	0.039141	0.037254	0.064605	0.016149

Date	USB	MS	KEY	PNC	COF
2020-03-02 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2020-03-03 05:00:00+00:00	-0.042774	-0.044750	-0.044785	-0.053905	-0.055209
2020-03-03 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2020-03-04 05:00:00+00:00	0.012148	0.018919	0.029611	0.031158	0.033673
2020-03-04 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
...	...	...	...	...	...
2020-04-27 04:00:00+00:00	0.055294	0.035291	0.065138	0.047255	0.054141
2020-04-27 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2020-04-28 04:00:00+00:00	0.021461	0.020860	0.018949	0.004740	0.077040
2020-04-28 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2020-04-29 04:00:00+00:00	0.049386	0.025390	0.035503	0.047174	0.094118

Date	...	CSCO	DOC	EQIX	DUK
2020-03-02 16:00:00+00:00	...	0.000000	0.000000	0.000000	0.000000
2020-03-03 05:00:00+00:00	...	-0.027447	0.012184	-0.006011	-0.010965
2020-03-03 16:00:00+00:00	...	0.000000	0.000000	0.000000	0.000000
2020-03-04 05:00:00+00:00	...	0.033716	0.040325	0.049304	0.063173
2020-03-04 16:00:00+00:00	...	0.000000	0.000000	0.000000	0.000000
...	...	...	...	...	...
2020-04-27 04:00:00+00:00	...	0.012700	0.044985	0.024668	0.012370
2020-04-27 16:00:00+00:00	...	0.000000	0.000000	0.000000	0.000000
2020-04-28 04:00:00+00:00	...	-0.013238	0.046998	-0.025983	0.010144
2020-04-28 16:00:00+00:00	...	0.000000	0.000000	0.000000	0.000000
2020-04-29 04:00:00+00:00	...	0.020005	0.030177	-0.006493	-0.011412

Date	NFLX	GE	APA	F	REGN
2020-03-02 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2020-03-03 05:00:00+00:00	-0.032227	-0.029438	-0.020768	-0.031944	-0.006885
2020-03-03 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2020-03-04 05:00:00+00:00	0.040730	0.006434	0.005602	0.015782	0.069180
2020-03-04 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
...	...	...	...	...	...
2020-04-27 04:00:00+00:00	-0.008494	0.027157	-0.017774	0.061602	-0.033009
2020-04-27 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2020-04-28 04:00:00+00:00	-0.041649	0.057543	0.016191	0.040619	-0.034538
2020-04-28 16:00:00+00:00	0.000000	0.000000	0.000000	0.000000	0.000000
2020-04-29 04:00:00+00:00	0.019959	-0.032353	0.223055	-0.022305	-0.023571

```
                                CMS
Date
2020-03-02 16:00:00+00:00  0.000000
2020-03-03 05:00:00+00:00  0.002806
2020-03-03 16:00:00+00:00  0.000000
2020-03-04 05:00:00+00:00  0.062655
2020-03-04 16:00:00+00:00  0.000000
...                             ...
2020-04-27 04:00:00+00:00 -0.023569
2020-04-27 16:00:00+00:00  0.000000
2020-04-28 04:00:00+00:00  0.020862
2020-04-28 16:00:00+00:00  0.000000
2020-04-29 04:00:00+00:00 -0.032934

[80 rows x 30 columns]
```

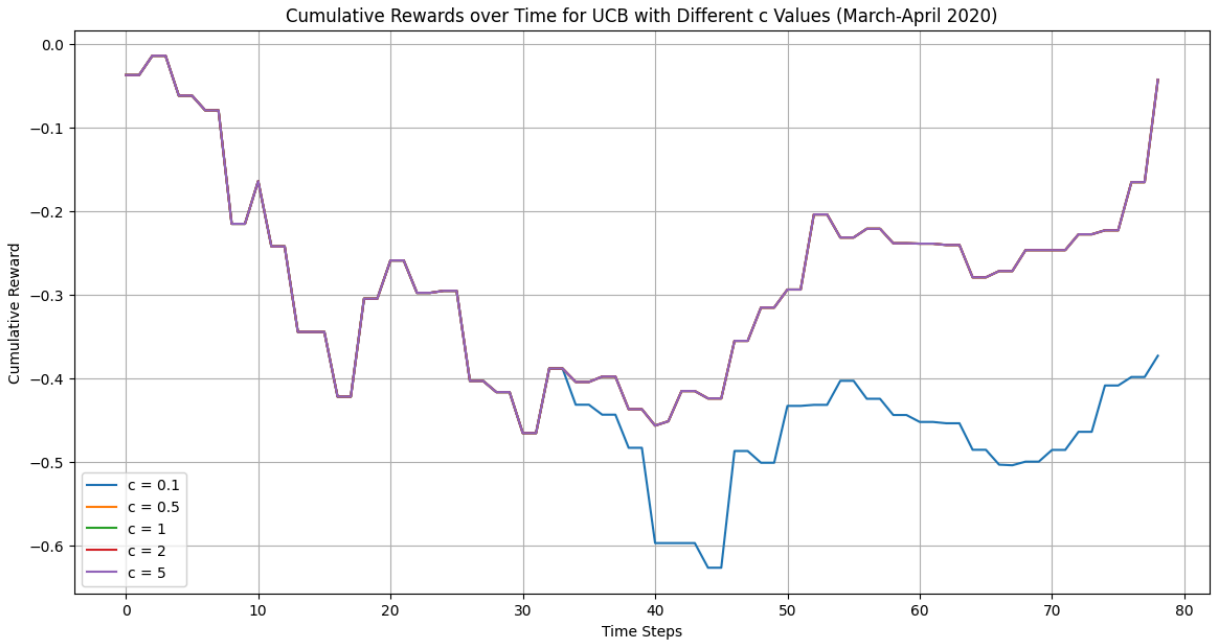
Step 11: Re-evaluation of Algorithms with Updated Data and Parameter Adjustments

Modify the UCB Algorithm for the Updated Dataset

In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Function to implement the UCB algorithm with Q-value calculation for updated data
def ucb_algorithm_with_q_values(data, c):
    num_assets = data.shape[1]  # Number of assets
    Q_values = np.zeros(num_assets)  # Estimated returns for each asset (Q-values)
    N_counts = np.zeros(num_assets)  # Count of selections for each asset
    reward_history = []  # Store the rewards for cumulative reward calculation

    num_steps = len(data)
    for t in range(1, num_steps):
        UCB_scores = np.zeros(num_assets)

        for a in range(num_assets):
            if N_counts[a] == 0:
                UCB_scores[a] = float('inf')
            else:
                UCB_scores[a] = Q_values[a] + c * np.sqrt(np.log(t) / N_counts[a])

        selected_asset = np.argmax(UCB_scores)  # Select asset with highest UCB sco
        reward = data.iloc[t, selected_asset]  # Get the reward for the selected as
        N_counts[selected_asset] += 1
        # Update Q-value for the selected asset using incremental formula
        Q_values[selected_asset] += (reward - Q_values[selected_asset]) / N_counts[
        reward_history.append(reward)

    return Q_values, reward_history
```

```
# Run the UCB algorithm with different c values
c_values = [0.1, 0.5, 1, 2, 5]
plt.figure(figsize=(14, 7))

for c in c_values:
    Q_values, reward_history = ucb_algorithm_with_q_values(data_combined_returns_up
    cumulative_rewards = np.cumsum(reward_history)

    print(f"\nResults for c = {c}")
    print("Final Q-values (estimated returns) for each asset:", Q_values)
    print("Total accumulated rewards:", sum(reward_history))

    # Plot cumulative rewards
    plt.plot(cumulative_rewards, label=f"c = {c}")

plt.title("Cumulative Rewards over Time for UCB with Different c Values (March-Apri
plt.xlabel("Time Steps")
plt.ylabel("Cumulative Reward")
plt.legend()
plt.grid(True)
plt.show()
```

```
Results for c = 0.1
Final Q-values (estimated returns) for each asset: [-0.01876228  0.01448677  0.00768
768 -0.01056106 -0.02797867 -0.01979365
  0.00259337  0.         -0.13551588 -0.05692189  0.00983149 -0.07793865
 -0.00592149 -0.10219844 -0.00024555  0.00145067 -0.07734635 -0.0147278
  0.02382243  0.          0.02528417  0.02789551 -0.02904525 -0.00129556
  0.00080313  0.00466558 -0.10766056  0.01698511 -0.00405279  0.        ]
Total accumulated rewards: -0.3729478763369485

Results for c = 0.5
Final Q-values (estimated returns) for each asset: [ 0.01739637  0.         0.00768
768  0.00213096 -0.03761829  0.00629306
 -0.00880488 -0.01946247 -0.06775794  0.          0.00409816 -0.0337108
 -0.00981553 -0.05109922  0.00174298  0.01452725 -0.03867318  0.00160077
  0.0221848   0.0378713   0.04080076  0.         -0.01934843  0.0229327
 -0.00688092  0.0191809  -0.0624936   0.01324503  0.01106746  0.        ]
Total accumulated rewards: -0.04304998106174851

Results for c = 1
Final Q-values (estimated returns) for each asset: [ 0.01739637  0.         0.00768
768  0.00213096 -0.03761829  0.00629306
 -0.00880488 -0.01946247 -0.06775794  0.          0.00409816 -0.0337108
 -0.00981553 -0.05109922  0.00174298  0.01452725 -0.03867318  0.00160077
  0.0221848   0.0378713   0.04080076  0.         -0.01934843  0.0229327
 -0.00688092  0.0191809  -0.0624936   0.01324503  0.01106746  0.        ]
Total accumulated rewards: -0.04304998106174851

Results for c = 2
Final Q-values (estimated returns) for each asset: [ 0.01739637  0.         0.00768
768  0.00213096 -0.03761829  0.00629306
 -0.00880488 -0.01946247 -0.06775794  0.          0.00409816 -0.0337108
 -0.00981553 -0.05109922  0.00174298  0.01452725 -0.03867318  0.00160077
  0.0221848   0.0378713   0.04080076  0.         -0.01934843  0.0229327
 -0.00688092  0.0191809  -0.0624936   0.01324503  0.01106746  0.        ]
Total accumulated rewards: -0.04304998106174851

Results for c = 5
Final Q-values (estimated returns) for each asset: [ 0.01739637  0.         0.00768
768  0.00213096 -0.03761829  0.00629306
 -0.00880488 -0.01946247 -0.06775794  0.          0.00409816 -0.0337108
 -0.00981553 -0.05109922  0.00174298  0.01452725 -0.03867318  0.00160077
  0.0221848   0.0378713   0.04080076  0.         -0.01934843  0.0229327
 -0.00688092  0.0191809  -0.0624936   0.01324503  0.01106746  0.        ]
Total accumulated rewards: -0.04304998106174851
```

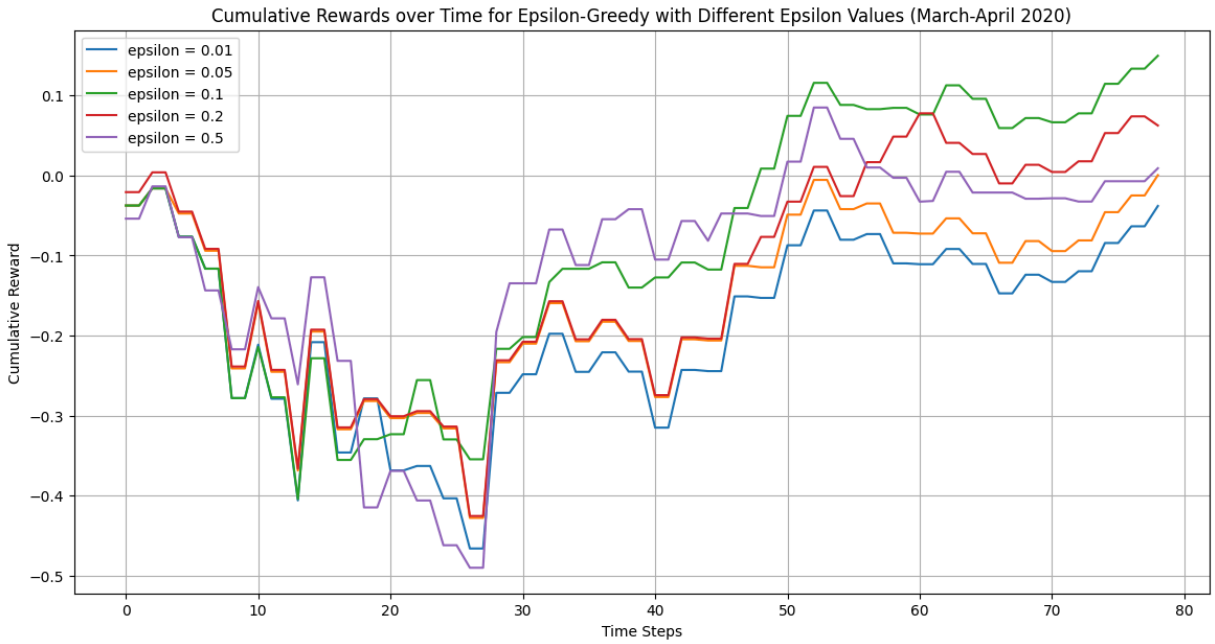


Modify the Epsilon-Greedy Algorithm for the Updated Dataset

In [ ]:
```
# Function to implement the epsilon-greedy algorithm with Q-value calculation for u
def epsilon_greedy_algorithm_with_q_values(data, epsilon):
    num_assets = data.shape[1]  # Number of assets
    Q_values = np.zeros(num_assets)  # Estimated returns for each asset (Q-values)
    N_counts = np.zeros(num_assets)  # Count of selections for each asset
    reward_history = []  # Store rewards for cumulative reward calculation

    num_steps = len(data)
    for t in range(1, num_steps):
        if np.random.rand() < epsilon:  # Exploration
            selected_asset = np.random.randint(0, num_assets)
        else:  # Exploitation
            selected_asset = np.argmax(Q_values)

        reward = data.iloc[t, selected_asset]  # Get the reward for the selected as
        N_counts[selected_asset] += 1
        # Update Q-value for the selected asset using incremental formula
        Q_values[selected_asset] += (reward - Q_values[selected_asset]) / N_counts[
        reward_history.append(reward)

    return Q_values, reward_history

# Run the epsilon-greedy algorithm with different epsilon values
epsilon_values = [0.01, 0.05, 0.1, 0.2, 0.5]
plt.figure(figsize=(14, 7))

for epsilon in epsilon_values:
    Q_values, reward_history = epsilon_greedy_algorithm_with_q_values(data_combined
    cumulative_rewards = np.cumsum(reward_history)

    print(f"\nResults for epsilon = {epsilon}")
    print("Final Q-values (estimated returns) for each asset:", Q_values)
    print("Total accumulated rewards:", sum(reward_history))
```

```
    # Plot cumulative rewards
    plt.plot(cumulative_rewards, label=f"epsilon = {epsilon}")

plt.title("Cumulative Rewards over Time for Epsilon-Greedy with Different Epsilon V
plt.xlabel("Time Steps")
plt.ylabel("Cumulative Reward")
plt.legend()
plt.grid(True)
plt.show()
```

```
Results for epsilon = 0.01
Final Q-values (estimated returns) for each asset: [-0.03752456 -0.00973022 -0.01997
768 -0.08085835 -0.00027006 -0.06351841
  0.00790202  0.          0.          0.         -0.13785138  0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.        ]
Total accumulated rewards: -0.03808632117119781

Results for epsilon = 0.05
Final Q-values (estimated returns) for each asset: [-0.03752456 -0.00501283 -0.07351
212 -0.00140925 -0.123393   -0.00454876
  0.0086185   0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.         -0.03143666  0.          0.          0.00484939]
Total accumulated rewards: 0.00043164139618456776

Results for epsilon = 0.1
Final Q-values (estimated returns) for each asset: [-0.01876228 -0.00973022 -0.01997
768 -0.07151013  0.00931728 -0.03151204
  0.          0.          0.          0.          0.          0.
  0.          0.          0.0082812   0.00630814  0.00310948  0.
  0.         -0.00881838  0.          0.          0.00735498  0.
  0.          0.          0.          0.          0.          0.        ]
Total accumulated rewards: 0.1496360647879863

Results for epsilon = 0.2
Final Q-values (estimated returns) for each asset: [-0.00608802 -0.02326476 -0.07351
212 -0.00140925 -0.0616965  -0.00492783
  0.01054085  0.          0.         -0.01391163  0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.00581606  0.          0.00843425  0.         -0.01141173
  0.00950496  0.          0.          0.          0.          0.        ]
Total accumulated rewards: 0.06248180045593377

Results for epsilon = 0.5
Final Q-values (estimated returns) for each asset: [-0.00839406  0.          0.
0.          0.01614855  0.
  0.          0.         -0.05390536 -0.04440977  0.          0.
 -0.0158772   0.          0.01366356  0.          0.01370253  0.01253356
  0.         -0.00969808  0.00488889 -0.01249454  0.          0.
 -0.01300817 -0.03296267  0.0151428   0.01430839  0.          0.        ]
Total accumulated rewards: 0.009013376295270992
```

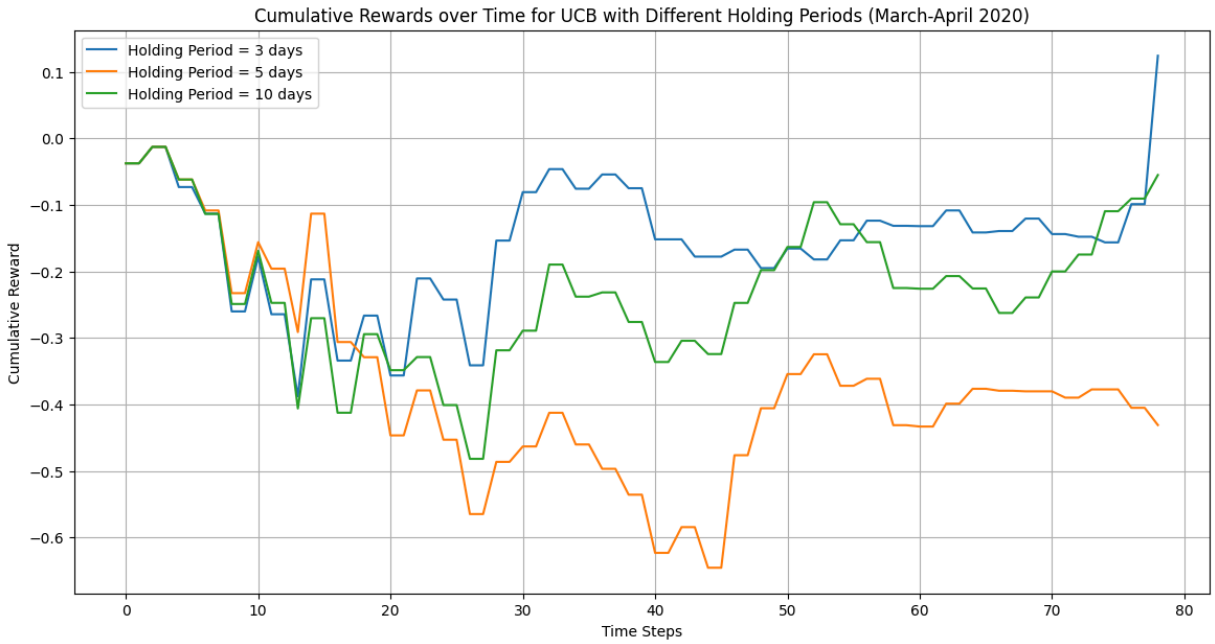


UCB with Holding Period

In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Function to implement the UCB algorithm with a holding period
def ucb_with_holding_period(data, c, holding_period):
    num_assets = data.shape[1]  # Number of assets
    Q_values = np.zeros(num_assets)  # Estimated returns for each asset
    N_counts = np.zeros(num_assets)  # Count of selections for each asset
    reward_history = []  # To store rewards for cumulative calculation
    num_steps = len(data)

    for t in range(1, num_steps, holding_period):  # Step by holding period
        UCB_scores = np.zeros(num_assets)

        for a in range(num_assets):
            if N_counts[a] == 0:
                UCB_scores[a] = float('inf')
            else:
                UCB_scores[a] = Q_values[a] + c * np.sqrt(np.log(t) / N_counts[a])

        selected_asset = np.argmax(UCB_scores)

        # Hold the selected asset for the defined holding period
        for i in range(holding_period):
            if t + i < num_steps:
                reward = data.iloc[t + i, selected_asset]
                N_counts[selected_asset] += 1
                Q_values[selected_asset] += (reward - Q_values[selected_asset]) / N
                reward_history.append(reward)

    return Q_values, reward_history

# Run the UCB algorithm with different holding periods
```

```
c_value = 2.0  # Example UCB exploration constant
holding_periods = [3, 5, 10]
plt.figure(figsize=(14, 7))

for holding_period in holding_periods:
    Q_values, reward_history = ucb_with_holding_period(data_combined_returns_update
    cumulative_rewards = np.cumsum(reward_history)

    print(f"\nResults for holding period = {holding_period} days")
    print("Final Q-values (estimated returns) for each asset:", Q_values)
    print("Total accumulated rewards:", sum(reward_history))

    # Plot cumulative rewards
    plt.plot(cumulative_rewards, label=f"Holding Period = {holding_period} days")

plt.title("Cumulative Rewards over Time for UCB with Different Holding Periods (Mar
plt.xlabel("Time Steps")
plt.ylabel("Cumulative Reward")
plt.legend()
plt.grid(True)
plt.show()
```

```
Results for holding period = 3 days
Final Q-values (estimated returns) for each asset: [-0.00427177 -0.02012876 -0.06232
654 -0.00140925  0.01747    -0.0407007
 -0.00745092  0.0486111  -0.04353225  0.06251453  0.03580504 -0.00982008
  0.00024679 -0.0256161  -0.0086614   0.0035099   0.00056181 -0.00547328
  0.01942157 -0.00258082  0.00771087 -0.01104344  0.00696362 -0.00776442
 -0.00416728  0.0191809   0.2230551   0.          0.          0.        ]
Total accumulated rewards: 0.1242024662826885

Results for holding period = 5 days
Final Q-values (estimated returns) for each asset: [-0.01237533 -0.03416441  0.02389
507 -0.04316142 -0.02480498 -0.00665882
  0.00523525 -0.0151064  -0.02196161  0.04795394  0.00675713 -0.01185322
  0.01094531 -0.00076227  0.00057664 -0.01340048  0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.        ]
Total accumulated rewards: -0.4310275250039851

Results for holding period = 10 days
Final Q-values (estimated returns) for each asset: [-0.02490121 -0.00453614 -0.00242
587  0.004258    0.0077907  -0.0026864
 -0.00141687  0.02045449  0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.        ]
Total accumulated rewards: -0.0550874877007822
```



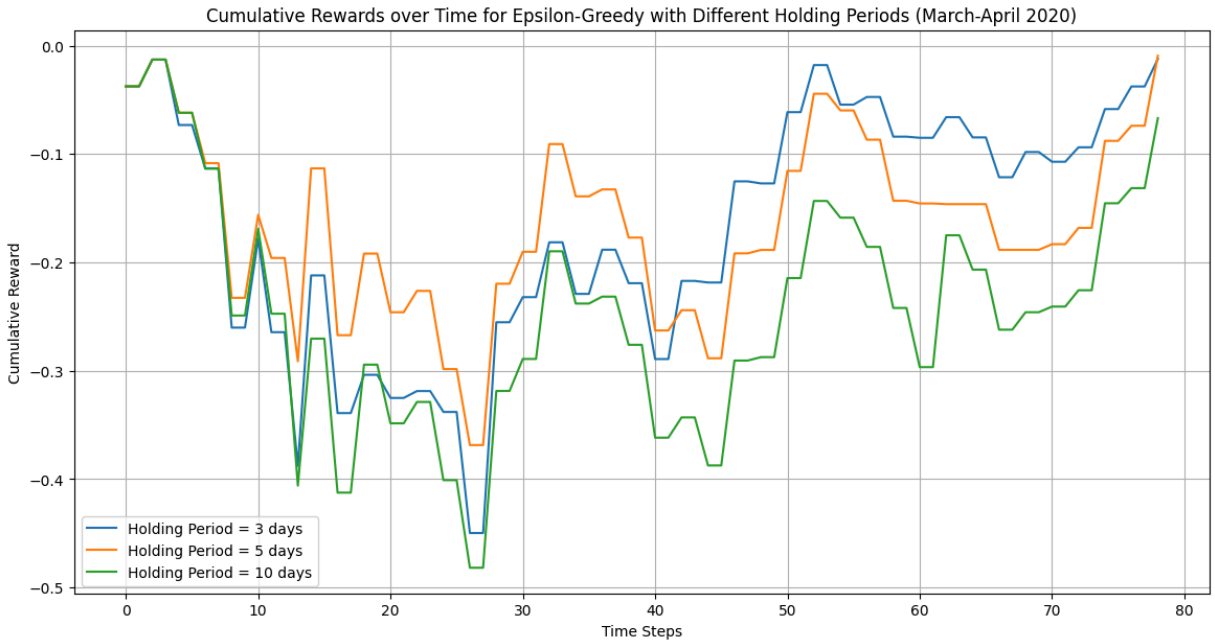
Epsilon-Greedy with Holding Period

In [ ]:

```
# Function to implement the epsilon-greedy algorithm with a holding period
def epsilon_greedy_with_holding_period(data, epsilon, holding_period):
    num_assets = data.shape[1]  # Number of assets
    Q_values = np.zeros(num_assets)  # Estimated returns for each asset
    N_counts = np.zeros(num_assets)  # Count of selections for each asset
    reward_history = []  # To store rewards for cumulative calculation
    num_steps = len(data)

    for t in range(1, num_steps, holding_period):  # Step by holding period
        if np.random.rand() < epsilon:  # Exploration
            selected_asset = np.random.randint(0, num_assets)
        else:  # Exploitation
            selected_asset = np.argmax(Q_values)

        # Hold the selected asset for the defined holding period
        for i in range(holding_period):
            if t + i < num_steps:
                reward = data.iloc[t + i, selected_asset]
                N_counts[selected_asset] += 1
                Q_values[selected_asset] += (reward - Q_values[selected_asset]) / N
                reward_history.append(reward)

    return Q_values, reward_history

# Run the epsilon-greedy algorithm with different holding periods
epsilon_value = 0.1  # Example epsilon value for exploration
holding_periods = [3, 5, 10]
plt.figure(figsize=(14, 7))

for holding_period in holding_periods:
    Q_values, reward_history = epsilon_greedy_with_holding_period(data_combined_ret
    cumulative_rewards = np.cumsum(reward_history)
```

```
    print(f"\nResults for holding period = {holding_period} days")
    print("Final Q-values (estimated returns) for each asset:", Q_values)
    print("Total accumulated rewards:", sum(reward_history))

    # Plot cumulative rewards
    plt.plot(cumulative_rewards, label=f"Holding Period = {holding_period} days")

plt.title("Cumulative Rewards over Time for Epsilon-Greedy with Different Holding P
plt.xlabel("Time Steps")
plt.ylabel("Cumulative Reward")
plt.legend()
plt.grid(True)
plt.show()
```

```
Results for holding period = 3 days
Final Q-values (estimated returns) for each asset: [-0.00427177 -0.02012876 -0.06232
654 -0.00140925 -0.01244051 -0.0123012
  0.00872808  0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.00328835
  0.          0.          0.          0.          0.          0.        ]
Total accumulated rewards: -0.012221943826652315

Results for holding period = 5 days
Final Q-values (estimated returns) for each asset: [-0.01237533 -0.03416441 -0.00438
298  0.00760169  0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.         -0.0006247   0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.         -0.00843375  0.          0.        ]
Total accumulated rewards: -0.009261002840913135

Results for holding period = 10 days
Final Q-values (estimated returns) for each asset: [-0.02490121 -0.00453614 -0.00242
587  0.00513835  0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.        ]
Total accumulated rewards: -0.06685313014845151
```

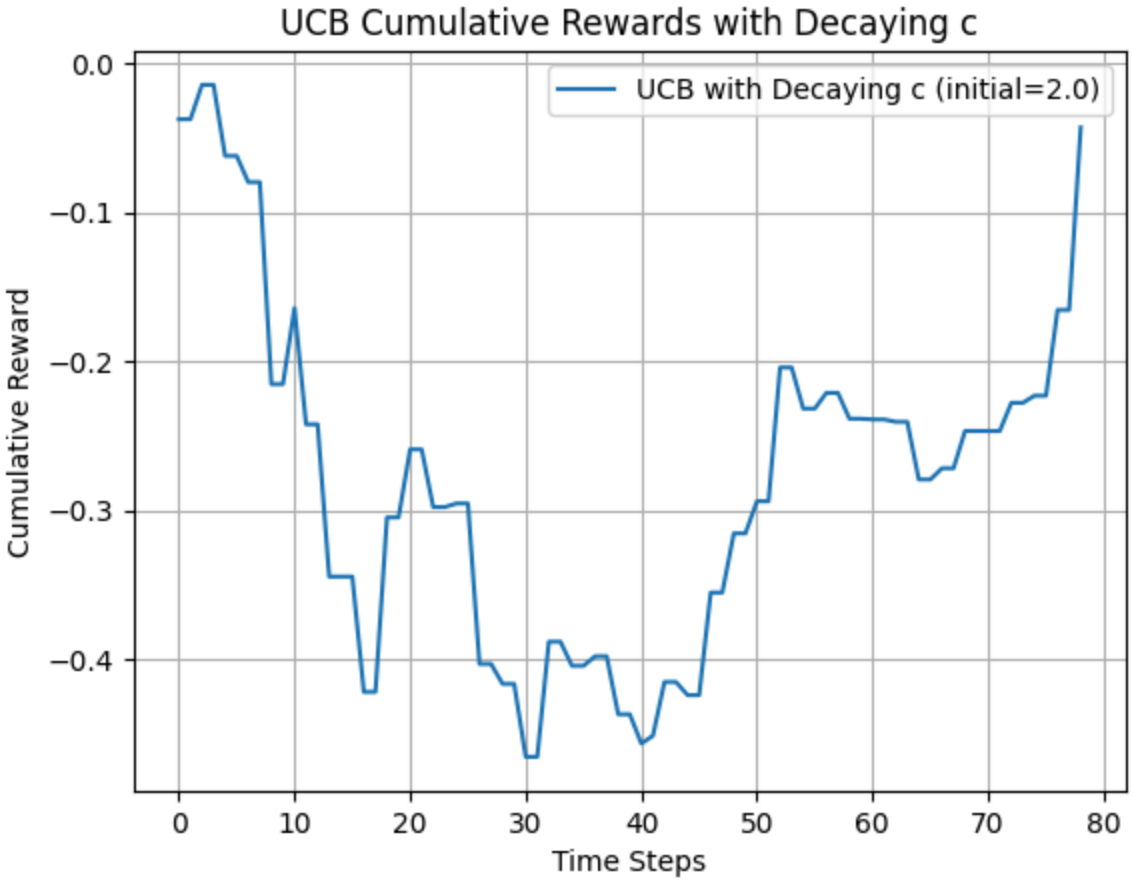


Decaying Exploration Parameters

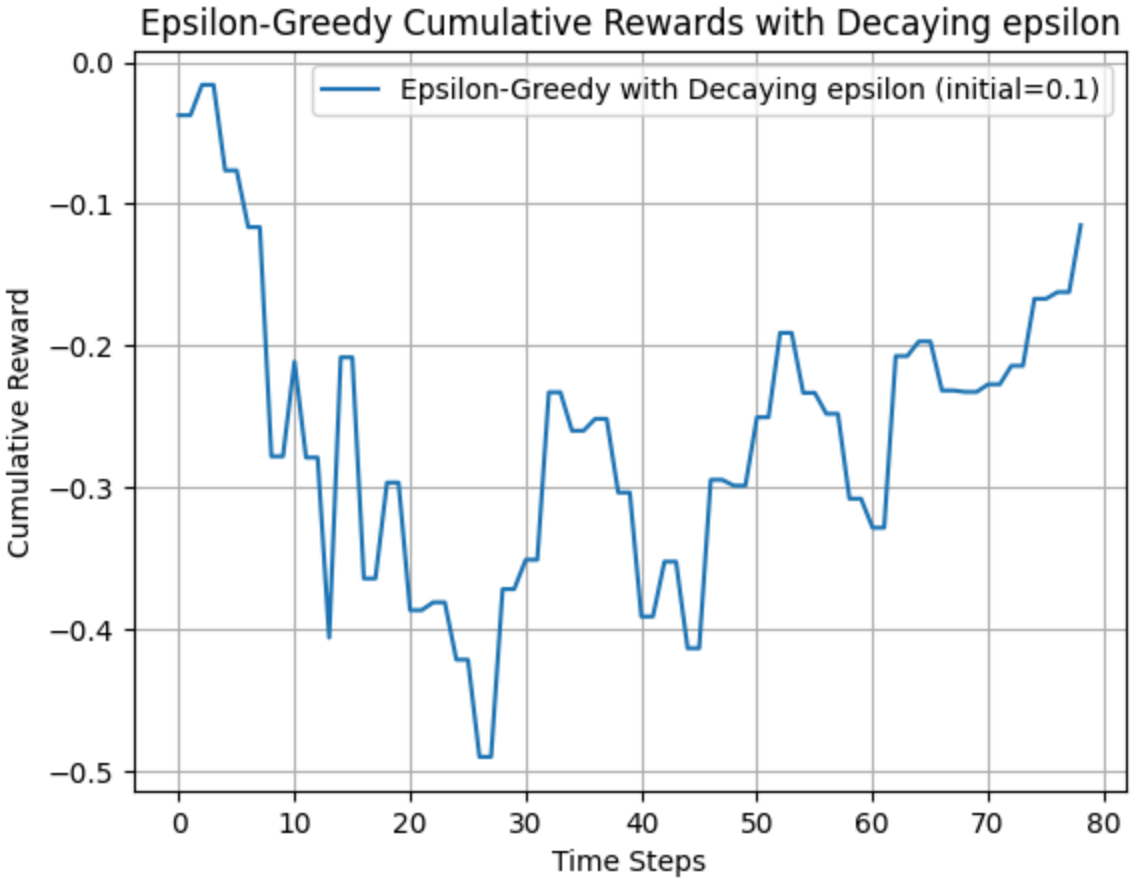
In [ ]:
```python
# UCB with Decaying c
def ucb_with_decay(data, initial_c=2.0, decay_rate=0.99):
    num_assets = data.shape[1]
    Q_values = np.zeros(num_assets)
    N_counts = np.zeros(num_assets)
    reward_history = []

    num_steps = len(data)
    for t in range(1, num_steps):
        c = initial_c * (decay_rate ** t)  # Decaying exploration constant
        UCB_scores = np.zeros(num_assets)
        for a in range(num_assets):
            if N_counts[a] == 0:
                UCB_scores[a] = float('inf')
            else:
                UCB_scores[a] = Q_values[a] + c * np.sqrt(np.log(t) / N_counts[a])

        selected_asset = np.argmax(UCB_scores)
        reward = data.iloc[t, selected_asset]
        N_counts[selected_asset] += 1
        Q_values[selected_asset] += (reward - Q_values[selected_asset]) / N_counts[
        reward_history.append(reward)

    cumulative_rewards = np.cumsum(reward_history)
    print("UCB with Decaying c - Final Q-values:", Q_values)
    print("Total accumulated rewards:", sum(reward_history))

    plt.plot(cumulative_rewards, label=f"UCB with Decaying c (initial={initial_c})"
    plt.title("UCB Cumulative Rewards with Decaying c")
    plt.xlabel("Time Steps")
    plt.ylabel("Cumulative Reward")
    plt.legend()
    plt.grid(True)
```

```
    plt.show()

# Epsilon-Greedy with Decaying epsilon
def epsilon_greedy_with_decay(data, initial_epsilon=0.1, decay_rate=0.99):
    num_assets = data.shape[1]
    Q_values = np.zeros(num_assets)
    N_counts = np.zeros(num_assets)
    reward_history = []

    num_steps = len(data)
    for t in range(1, num_steps):
        epsilon = initial_epsilon * (decay_rate ** t)
        if np.random.rand() < epsilon:
            selected_asset = np.random.randint(0, num_assets)
        else:
            selected_asset = np.argmax(Q_values)

        reward = data.iloc[t, selected_asset]
        N_counts[selected_asset] += 1
        Q_values[selected_asset] += (reward - Q_values[selected_asset]) / N_counts[
        reward_history.append(reward)

    cumulative_rewards = np.cumsum(reward_history)
    print("Epsilon-Greedy with Decaying epsilon - Final Q-values:", Q_values)
    print("Total accumulated rewards:", sum(reward_history))

    plt.plot(cumulative_rewards, label=f"Epsilon-Greedy with Decaying epsilon (init
    plt.title("Epsilon-Greedy Cumulative Rewards with Decaying epsilon")
    plt.xlabel("Time Steps")
    plt.ylabel("Cumulative Reward")
    plt.legend()
    plt.grid(True)
    plt.show()

# Run both decaying exploration versions
ucb_with_decay(data_combined_returns_updated)
epsilon_greedy_with_decay(data_combined_returns_updated)
```

```
UCB with Decaying c - Final Q-values: [ 0.01739637  0.          0.00768768  0.002130
96 -0.03761829  0.00629306
 -0.00880488 -0.01946247 -0.06775794  0.          0.00409816 -0.0337108
 -0.00981553 -0.05109922  0.00174298  0.01452725 -0.03867318  0.00160077
  0.0221848   0.0378713   0.04080076  0.         -0.01934843  0.0229327
 -0.00688092  0.0191809  -0.0624936   0.01324503  0.01106746  0.        ]
Total accumulated rewards: -0.04304998106174851
```




```
Epsilon-Greedy with Decaying epsilon - Final Q-values: [-0.03752456 -0.00973022 -0.0
1997768 -0.16171671 -0.00027006 -0.06351841
 -0.00141372 -0.03427624  0.00721166  0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.        ]
Total accumulated rewards: -0.11506138395886423
```

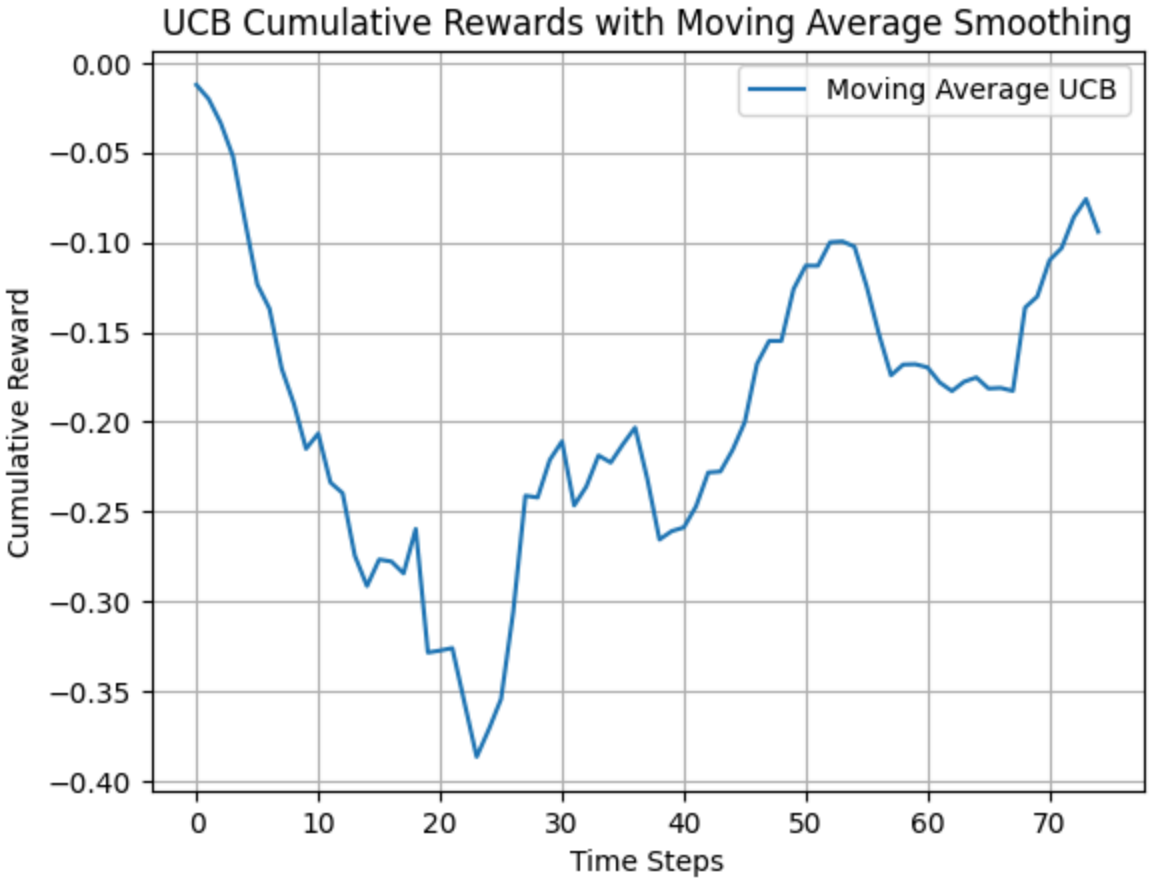


Incorporating Reward Smoothing (Moving Average)

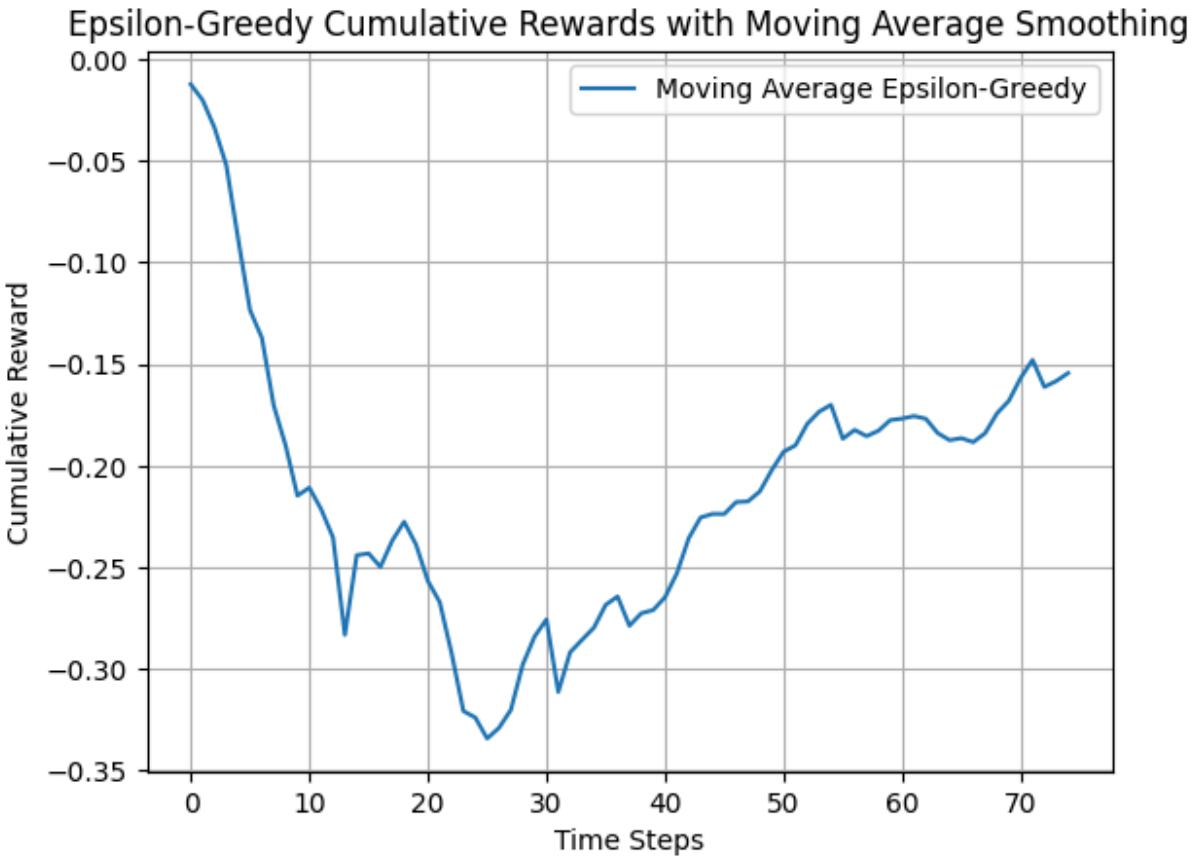
In [ ]:
```
# UCB with Moving Average Reward Smoothing
def ucb_with_moving_average(data, c=2.0, window=5):
    num_assets = data.shape[1]
    Q_values = np.zeros(num_assets)
    N_counts = np.zeros(num_assets)
    reward_history = []

    smoothed_data = data.rolling(window=window).mean().dropna()

    num_steps = len(smoothed_data)
    for t in range(1, num_steps):
        UCB_scores = np.zeros(num_assets)
        for a in range(num_assets):
            if N_counts[a] == 0:
                UCB_scores[a] = float('inf')
            else:
                UCB_scores[a] = Q_values[a] + c * np.sqrt(np.log(t) / N_counts[a])

        selected_asset = np.argmax(UCB_scores)
        reward = smoothed_data.iloc[t, selected_asset]
        N_counts[selected_asset] += 1
        Q_values[selected_asset] += (reward - Q_values[selected_asset]) / N_counts[
        reward_history.append(reward)

    cumulative_rewards = np.cumsum(reward_history)
    print("UCB with Moving Average Smoothing - Final Q-values:", Q_values)
```

```
    print("Total accumulated rewards:", sum(reward_history))

    plt.plot(cumulative_rewards, label="Moving Average UCB")
    plt.title("UCB Cumulative Rewards with Moving Average Smoothing")
    plt.xlabel("Time Steps")
    plt.ylabel("Cumulative Reward")
    plt.legend()
    plt.grid(True)
    plt.show()

# Epsilon-Greedy with Moving Average Reward Smoothing
def epsilon_greedy_with_moving_average(data, epsilon=0.1, window=5):
    num_assets = data.shape[1]
    Q_values = np.zeros(num_assets)
    N_counts = np.zeros(num_assets)
    reward_history = []

    smoothed_data = data.rolling(window=window).mean().dropna()

    num_steps = len(smoothed_data)
    for t in range(1, num_steps):
        if np.random.rand() < epsilon:
            selected_asset = np.random.randint(0, num_assets)
        else:
            selected_asset = np.argmax(Q_values)

        reward = smoothed_data.iloc[t, selected_asset]
        N_counts[selected_asset] += 1
        Q_values[selected_asset] += (reward - Q_values[selected_asset]) / N_counts[
        reward_history.append(reward)

    cumulative_rewards = np.cumsum(reward_history)
    print("Epsilon-Greedy with Moving Average Smoothing - Final Q-values:", Q_value
    print("Total accumulated rewards:", sum(reward_history))

    plt.plot(cumulative_rewards, label="Moving Average Epsilon-Greedy")
    plt.title("Epsilon-Greedy Cumulative Rewards with Moving Average Smoothing")
    plt.xlabel("Time Steps")
    plt.ylabel("Cumulative Reward")
    plt.legend()
    plt.grid(True)
    plt.show()

# Run both moving average versions
ucb_with_moving_average(data_combined_returns_updated)
epsilon_greedy_with_moving_average(data_combined_returns_updated)
```

```
UCB with Moving Average Smoothing - Final Q-values: [ 0.00439101  0.00715588  0.0042
4788  0.01013458 -0.01516933 -0.02902835
 -0.00041141 -0.02773475 -0.00324589 -0.01255521 -0.00999357 -0.00708664
  0.00374281 -0.03055733 -0.00848568  0.00879868  0.00560826 -0.00302996
  0.01019021 -0.03432837  0.00421098 -0.01623108 -0.01665296 -0.01483796
  0.01024306  0.00441884  0.01974438  0.02439932 -0.00570531  0.00992344]
Total accumulated rewards: -0.09418490003751712
```




```
Epsilon-Greedy with Moving Average Smoothing - Final Q-values: [-0.01237533 -0.00778
418 -0.01351667 -0.01760357 -0.03629254 -0.03495688
 -0.01360099 -0.0333018  -0.01958011 -0.00332988 -0.01394103 -0.04784874
 -0.00100006 -0.02849425 -0.00319288 -0.00139204  0.00618909  0.
  0.00206969  0.         -0.01041968  0.          0.         -0.01436758
  0.00411604  0.         -0.03566421  0.         -0.01326529  0.        ]
Total accumulated rewards: -0.15443549328090667
```

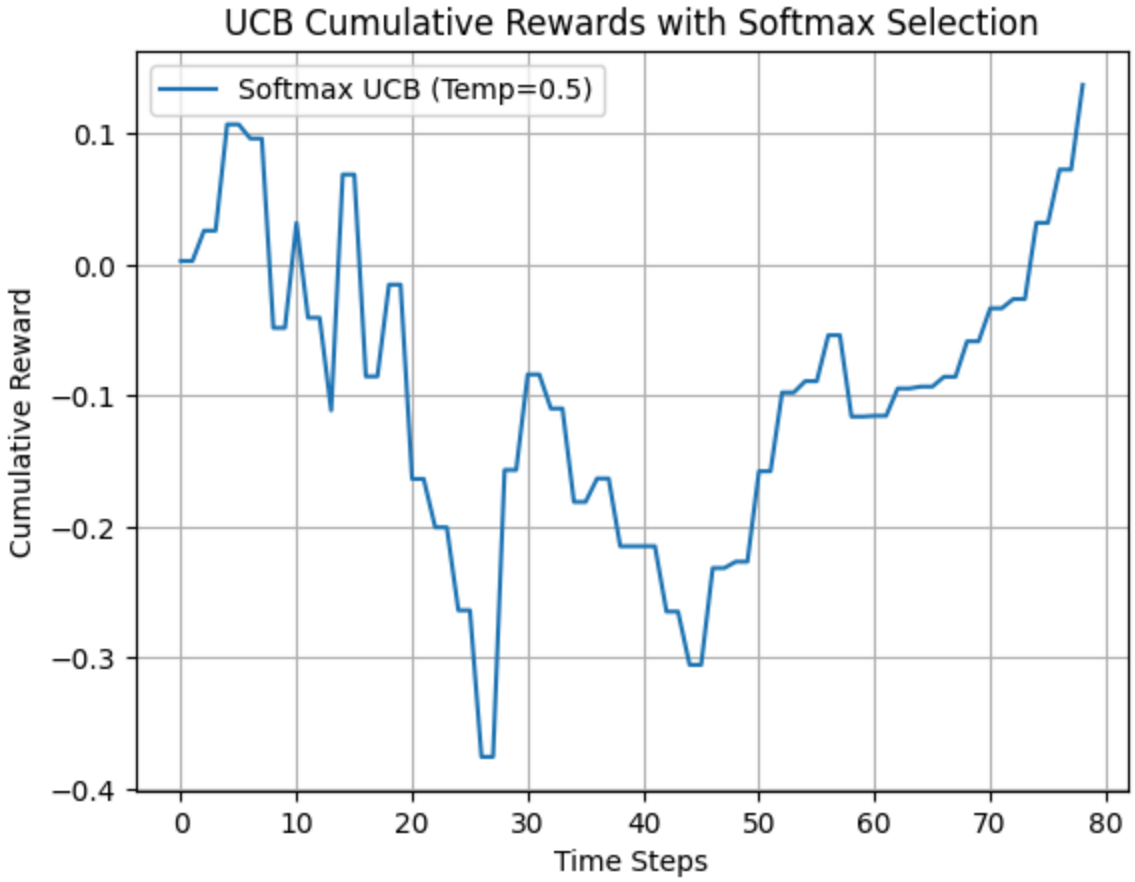


Alternative Policies (Softmax Selection with Temperature)

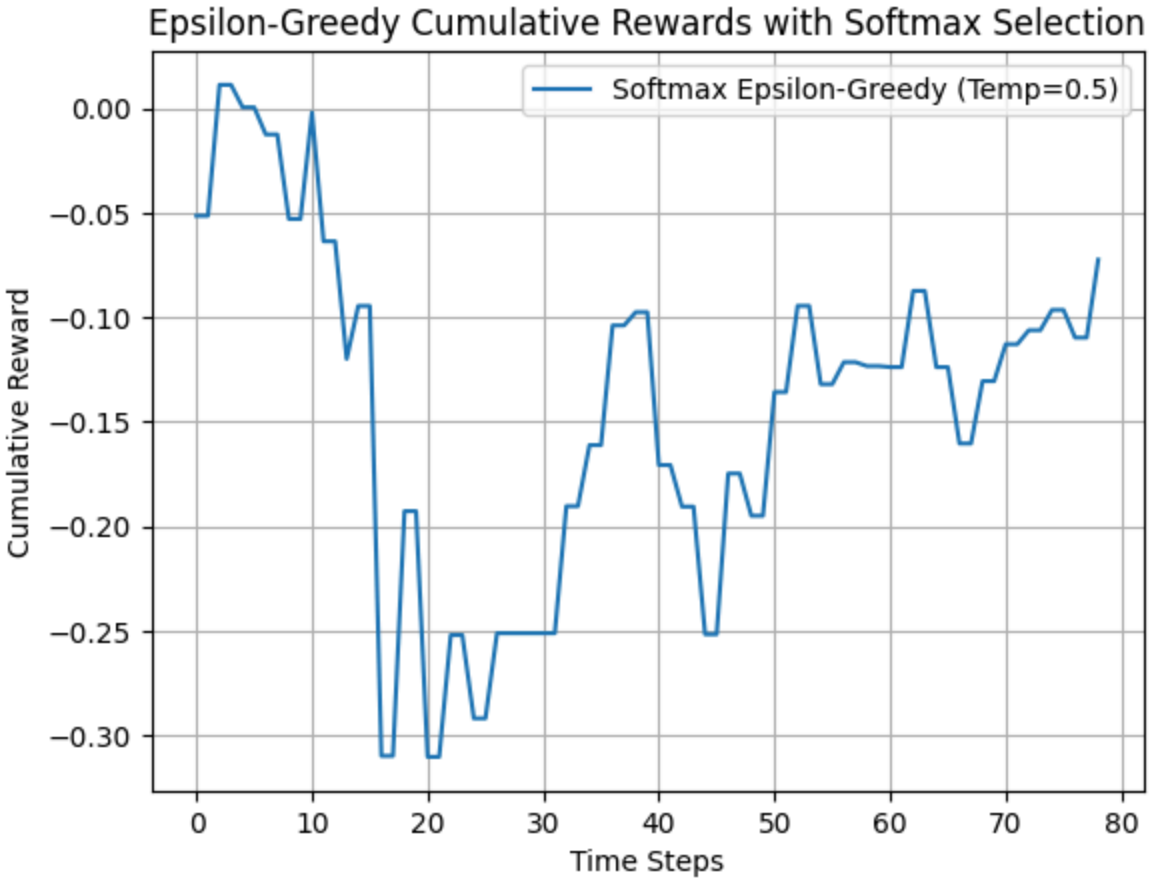
In [ ]:
```
# Softmax Selection for UCB
def softmax_ucb(data, temperature=1.0):
    num_assets = data.shape[1]
    Q_values = np.zeros(num_assets)
    N_counts = np.zeros(num_assets)
    reward_history = []

    num_steps = len(data)
    for t in range(1, num_steps):
        exp_values = np.exp(Q_values / temperature)
        probabilities = exp_values / np.sum(exp_values)
        selected_asset = np.random.choice(range(num_assets), p=probabilities)

        reward = data.iloc[t, selected_asset]
        N_counts[selected_asset] += 1
        Q_values[selected_asset] += (reward - Q_values[selected_asset]) / N_counts[
        reward_history.append(reward)

    cumulative_rewards = np.cumsum(reward_history)
    print("Softmax UCB - Final Q-values:", Q_values)
    print("Total accumulated rewards:", sum(reward_history))

    plt.plot(cumulative_rewards, label=f"Softmax UCB (Temp={temperature})")
    plt.title("UCB Cumulative Rewards with Softmax Selection")
    plt.xlabel("Time Steps")
    plt.ylabel("Cumulative Reward")
```

```
    plt.legend()
    plt.grid(True)
    plt.show()

# Softmax Selection for Epsilon-Greedy
def softmax_epsilon_greedy(data, temperature=1.0):
    num_assets = data.shape[1]
    Q_values = np.zeros(num_assets)
    N_counts = np.zeros(num_assets)
    reward_history = []

    num_steps = len(data)
    for t in range(1, num_steps):
        exp_values = np.exp(Q_values / temperature)
        probabilities = exp_values / np.sum(exp_values)
        selected_asset = np.random.choice(range(num_assets), p=probabilities)

        reward = data.iloc[t, selected_asset]
        N_counts[selected_asset] += 1
        Q_values[selected_asset] += (reward - Q_values[selected_asset]) / N_counts[
        reward_history.append(reward)

    cumulative_rewards = np.cumsum(reward_history)
    print("Softmax Epsilon-Greedy - Final Q-values:", Q_values)
    print("Total accumulated rewards:", sum(reward_history))

    plt.plot(cumulative_rewards, label=f"Softmax Epsilon-Greedy (Temp={temperature}
    plt.title("Epsilon-Greedy Cumulative Rewards with Softmax Selection")
    plt.xlabel("Time Steps")
    plt.ylabel("Cumulative Reward")
    plt.legend()
    plt.grid(True)
    plt.show()

# Run both softmax versions
softmax_ucb(data_combined_returns_updated, temperature=0.5)
softmax_epsilon_greedy(data_combined_returns_updated, temperature=0.5)
```

```
Softmax UCB - Final Q-values: [-0.02374644  0.00722218 -0.01212506  0.08148258  0.
-0.08538826
  0.          0.06001769  0.          0.          0.02295664  0.
  0.07381008  0.03448842  0.          0.02878309  0.          0.
  0.00353997 -0.01661782  0.00070625 -0.01846515  0.          0.00413545
  0.02332352 -0.03359133  0.          0.00995064 -0.07060468  0.0025194 ]
Total accumulated rewards: 0.13712537510587353
```




```
Softmax Epsilon-Greedy - Final Q-values: [-3.64573513e-02  0.00000000e+00  4.8211517
5e-02 -3.98585461e-02
  7.46715625e-05 -3.65747971e-02  0.00000000e+00 -6.31771082e-02
 -3.05301751e-02  0.00000000e+00 -5.14623559e-02  3.50568412e-03
  0.00000000e+00  5.82627235e-02  0.00000000e+00  6.83754145e-03
  1.90937689e-02  2.52822297e-02  2.88061198e-02 -5.20746292e-03
  5.37341546e-03  2.91940274e-02 -2.10873092e-03  1.33930313e-03
  7.67449219e-03 -3.38732837e-03  0.00000000e+00  0.00000000e+00
 -1.02381370e-02  6.26553203e-02]
Total accumulated rewards: -0.07247363241253424
```



References

Auer, Peter. "Using Confidence Bounds for Exploitation-Exploration Trade-Offs." Journal of Machine Learning Research, vol. 3, 2002, pp. 397–422.

Bubeck, Sébastien, and Nicolò Cesa-Bianchi. Regret Analysis of Stochastic and Nonstochastic Multi-Armed Bandit Problems. 2, arXiv, 2012. DOI.org (Datacite), https://doi.org/10.48550/ARXIV.1204.5721.

Cesa-Bianchi, Nicolo, and Gabor Lugosi. Prediction, Learning, and Games. 1st ed., Cambridge University Press, 2006. DOI.org (Crossref), https://doi.org/10.1017/CBO9780511546921.

Huo, Xiaoguang, and Feng Fu. "Risk-Aware Multi-Armed Bandit Problem with Application to Portfolio Selection." Royal Society Open Science, vol. 4, no. 11, Nov. 2017, p. 171377. DOI.org (Crossref), https://doi.org/10.1098/rsos.171377.

Lai, T. L., and Herbert Robbins. "Asymptotically Efficient Adaptive Allocation Rules." Advances in Applied Mathematics, vol. 6, no. 1, Mar. 1985, pp. 4–22. DOI.org (Crossref), https://doi.org/10.1016/0196-8858(85)90002-8.

Lattimore, Tor, and Csaba Szepesvári. Bandit Algorithms. 1st ed., Cambridge University Press, 2020. DOI.org (Crossref), https://doi.org/10.1017/9781108571401.

Li, Bin, and Steven C. H. Hoi. “Online Portfolio Selection: A Survey.” ACM Computing Surveys, vol. 46, no. 3, Jan. 2014, pp. 1–36. DOI.org (Crossref), https://doi.org/10.1145/2512962.

Markowitz, Harry. “PORTFOLIO SELECTION*.” The Journal of Finance, vol. 7, no. 1, Mar. 1952, pp. 77–91. DOI.org (Crossref), https://doi.org/10.1111/j.1540-6261.1952.tb01525.x.

Shen, Weiwei, et al. Portfolio Choices with Orthogonal Bandit Learning. 2015.

Shipra Agrawal and Navin Goyal. “Thompson Sampling for Contextual Bandits with Linear Payoffs.” Proceedings of the 30th International Conference on Machine Learning, edited by Sanjoy Dasgupta and David McAllester, vol. 28, no. 3, PMLR, 2013, pp. 127–35. Proceedings of Machine Learning Research, https://proceedings.mlr.press/v28/agrawal13.html.

Sutton, Richard S., and Andrew Barto. Reinforcement Learning: An Introduction. Second edition, The MIT Press, 2018.