

✓ MScFE 622 STOCHASTIC MODELING

Group Work Project # 3

Introduction:

This group project aims to apply reinforcement learning techniques to solve the portfolio selection problem. Inspired by Huo's research on risk-aware multi-armed bandit problems in portfolio selection, the project involves steps like selective reading, data collection, algorithm implementation, and performance comparison with Huo's results. By leveraging concepts like the Upper-Confidence Bound and epsilon-greedy algorithms, the team seeks to optimize portfolio selection using historical financial data. The ultimate objective is to evaluate the efficacy of these algorithms, particularly when applied to more recent data, thereby enhancing understanding of algorithmic trading strategies in real-world investment scenarios.

We will begin the task by loading the necessary Libraries:

✓ Loading the required Librarie

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

import yfinance as yf
from numpy.random import seed
from numpy.random import rand
from scipy.cluster import hierarchy
```

[link text](https:// [link text](https:// [link text](https:// [link text](#))))### Downloading and Exploring the Securities

At this phase of the project, team members are assigned individual tasks to progress towards applying reinforcement learning in portfolio selection. The task requires collaborative description of the portfolio selection problem as a multi-armed bandit problem and collecting sample data from Sep 2008 and Oct 2008, which includes gathering data for 15 financial institutions (JPM, WFC, BAC, C, GS, USB, MS, KEY, PNC, COF, AXP, PRU, SCHW, AFL, ALL) and 15 non-financial institutions (KR, PFE, XOM, WMT, DAL, CSCO, BKR, EQIX, DUK, NFLX, GE, APA, F, REGN, CMS). We then structured this data into a Python time series format, facilitating subsequent analysis and algorithm implementation. These initial steps lay the groundwork for the application of reinforcement learning techniques in portfolio optimization.

```

tickers = ["JPM", "WFC", "BAC", "C", "GS", "USB", "MS", "KEY", "PNC", "COF", "AXP",
           "PRU", "SCHW", "AFL", "ALL", "KR", "PFE", "XOM", "WMT", "DAL", "CSCO", "BKR",
           "EQIX", "DUK", "NFLX", "GE", "APA", "F", "REGN", "CMS"
          ]

len(tickers)

start_date = "2008-09-01"
end_date = "2008-10-31"

df=pd.DataFrame()
df_ret=pd.DataFrame()
for tick in tickers:
    price=yf.download(tick, start=start_date, end=end_date)
    plt.plot(price["Adj Close"] / price["Adj Close"][0], label=tick)
    price=price.rename(columns={"Adj Close" : tick})
    price[tick+"ret"]=price[tick].pct_change()
    df=pd.concat([df, price[[tick]]], axis=1)
    df_ret=pd.concat([df, price[[tick+"ret"]]], axis=1)
pdata = df.to_numpy()
pdata_dates = pd.to_datetime(price.index, format = "%Y-%m-%d")
legend = plt.legend(loc="lower left", shadow=True, fontsize="small", ncol=5)
#legend_title = plt.gca().add_artist(plt.Text(0, 0, "Stock Tickers", fontsize="small", fo
title="cumulative return"
plt.title(title, fontsize="x-large")
fig=plt.gcf()
fig.set_size_inches(15,8)
plt.show()

```

[illegible]

[illegible]

5 rows × 30 columns

✓ Portfolio Selection

In this task, we computed a 30 by 30 correlation matrix based on the daily returns of the downloaded securities. The correlation matrix serves as a fundamental tool for understanding the relationships between different securities within the portfolio. Additionally, we generated a heatmap visualization of the correlation matrix, providing a clear and intuitive representation of the correlations among the securities.

To facilitate portfolio management and decision-making, we decided to sort the securities based on their correlations. we opted for a hierarchical clustering approach, which organizes the securities into distinct groups based on their correlation patterns. By clustering securities with similar correlation profiles together, the group aims to create subsets of securities that exhibit similar behavior in the market. This approach enhances portfolio diversification and risk management by ensuring that assets with similar risk exposures are grouped together, allowing for more efficient allocation strategies. Ultimately, the chosen criteria for sorting the 30 stocks revolve around maximizing diversification benefits while minimizing potential systemic risks within the portfolio.

```
ret_cor = df.corr()  
ret_cor
```



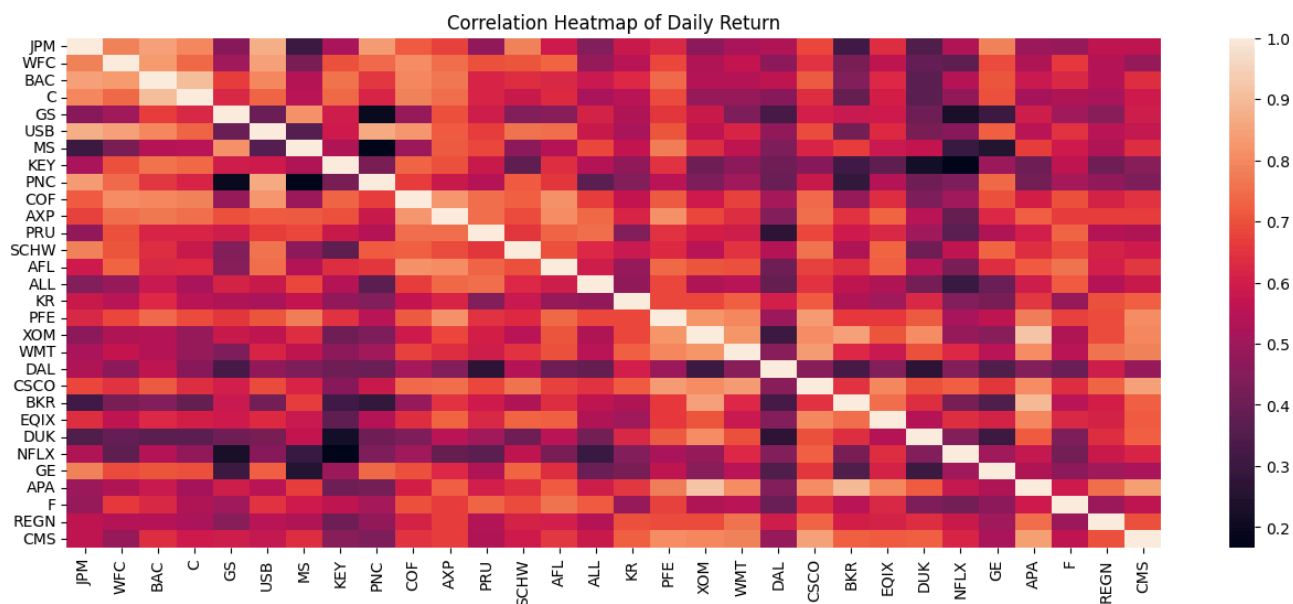
	JPM	WFC	BAC	C	GS	USB	MS	KEY
JPM	1.000000	0.788109	0.840628	0.800969	0.457663	0.874266	0.305244	0.516240
WFC	0.788109	1.000000	0.830237	0.736664	0.507067	0.843000	0.429167	0.695693
BAC	0.840628	0.830237	1.000000	0.904035	0.661774	0.795647	0.539035	0.753169
C	0.800969	0.736664	0.904035	1.000000	0.623447	0.735773	0.550132	0.742450
GS	0.457663	0.507067	0.661774	0.623447	1.000000	0.395573	0.814933	0.595028
USB	0.874266	0.843000	0.795647	0.735773	0.395573	1.000000	0.359952	0.589449
MS	0.305244	0.429167	0.539035	0.550132	0.814933	0.359952	1.000000	0.531887
KEY	0.516240	0.695693	0.753169	0.742450	0.595028	0.589449	0.531887	1.000000
PNC	0.831849	0.745078	0.653550	0.616419	0.193895	0.864231	0.168171	0.424826
COF	0.719847	0.805570	0.799047	0.790940	0.487569	0.821942	0.492566	0.730262
AXP	0.676410	0.747506	0.763410	0.748182	0.695820	0.717221	0.711553	0.702464
PRU	0.478625	0.702991	0.618638	0.615755	0.595896	0.669307	0.679493	0.585881
SCHW	0.785205	0.707921	0.640607	0.584780	0.447185	0.759740	0.461798	0.376911
AFL	0.590524	0.727467	0.619690	0.630457	0.453743	0.751178	0.536012	0.636889
ALL	0.444594	0.488070	0.579900	0.520852	0.610765	0.584094	0.680439	0.540216
KR	0.584293	0.552958	0.626818	0.549759	0.527767	0.517723	0.568998	0.477901
PFE	0.620028	0.683808	0.744738	0.691016	0.656893	0.705248	0.772369	0.647275
XOM	0.460087	0.527575	0.535174	0.488330	0.585311	0.562321	0.639848	0.409046
WMT	0.520155	0.570001	0.543592	0.480745	0.438044	0.614753	0.558746	0.461172
DAL	0.526229	0.466344	0.564399	0.458827	0.320428	0.475460	0.434232	0.409929
CSCO	0.682543	0.648629	0.710793	0.636118	0.600028	0.691789	0.618104	0.459450
BKR	0.317198	0.429354	0.449557	0.383236	0.583256	0.420247	0.669138	0.313700
EQIX	0.637278	0.564222	0.628118	0.607205	0.587596	0.629265	0.577915	0.373145
DUK	0.343695	0.390886	0.365083	0.365735	0.404720	0.420875	0.572371	0.218169
NFLX	0.529818	0.378512	0.538078	0.474861	0.226108	0.459025	0.296235	0.167087
GE	0.782299	0.689847	0.704273	0.697408	0.308085	0.721771	0.246726	0.494728
APA	0.494772	0.529243	0.579602	0.514996	0.597169	0.552584	0.666600	0.407390
F	0.486592	0.659980	0.623456	0.532483	0.500632	0.646822	0.590733	0.564468
REGN	0.558347	0.541780	0.541941	0.518864	0.453065	0.546532	0.533809	0.405321
CMS	0.567143	0.482438	0.637232	0.591420	0.594519	0.575621	0.637429	0.456178

```
plt.figure(figsize=(16, 6))

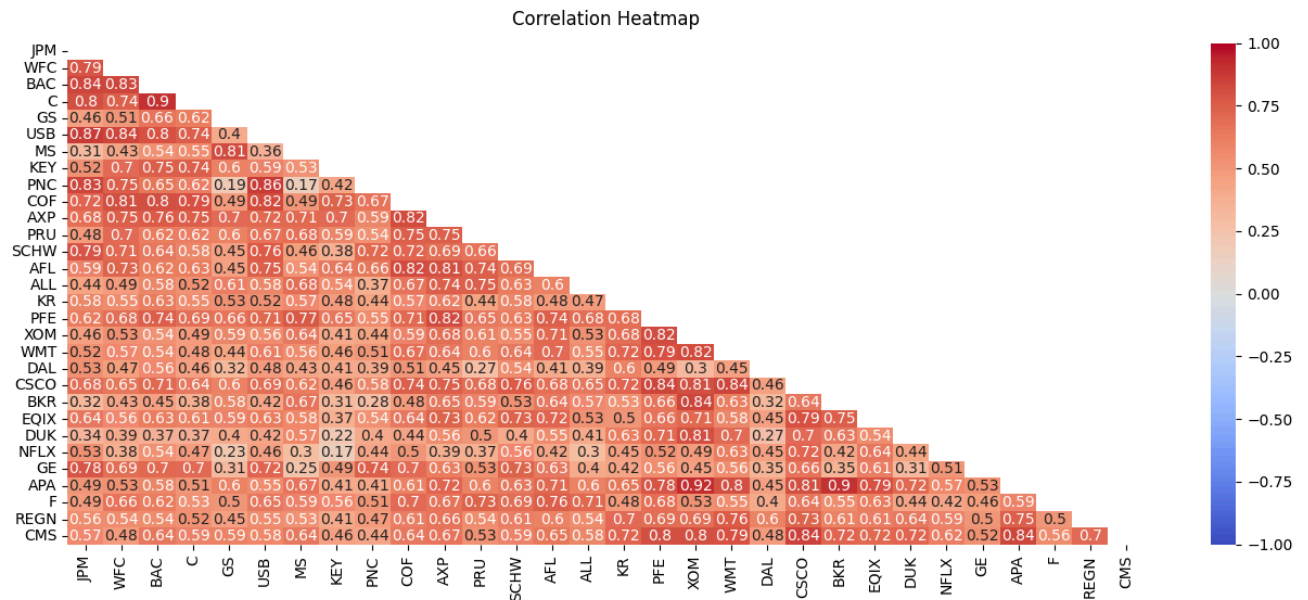
# Create the heatmap plot
sns.heatmap(ret_cor)

# Set the title
plt.title("Correlation Heatmap of Daily Return")

# Show the plot
plt.show()
```


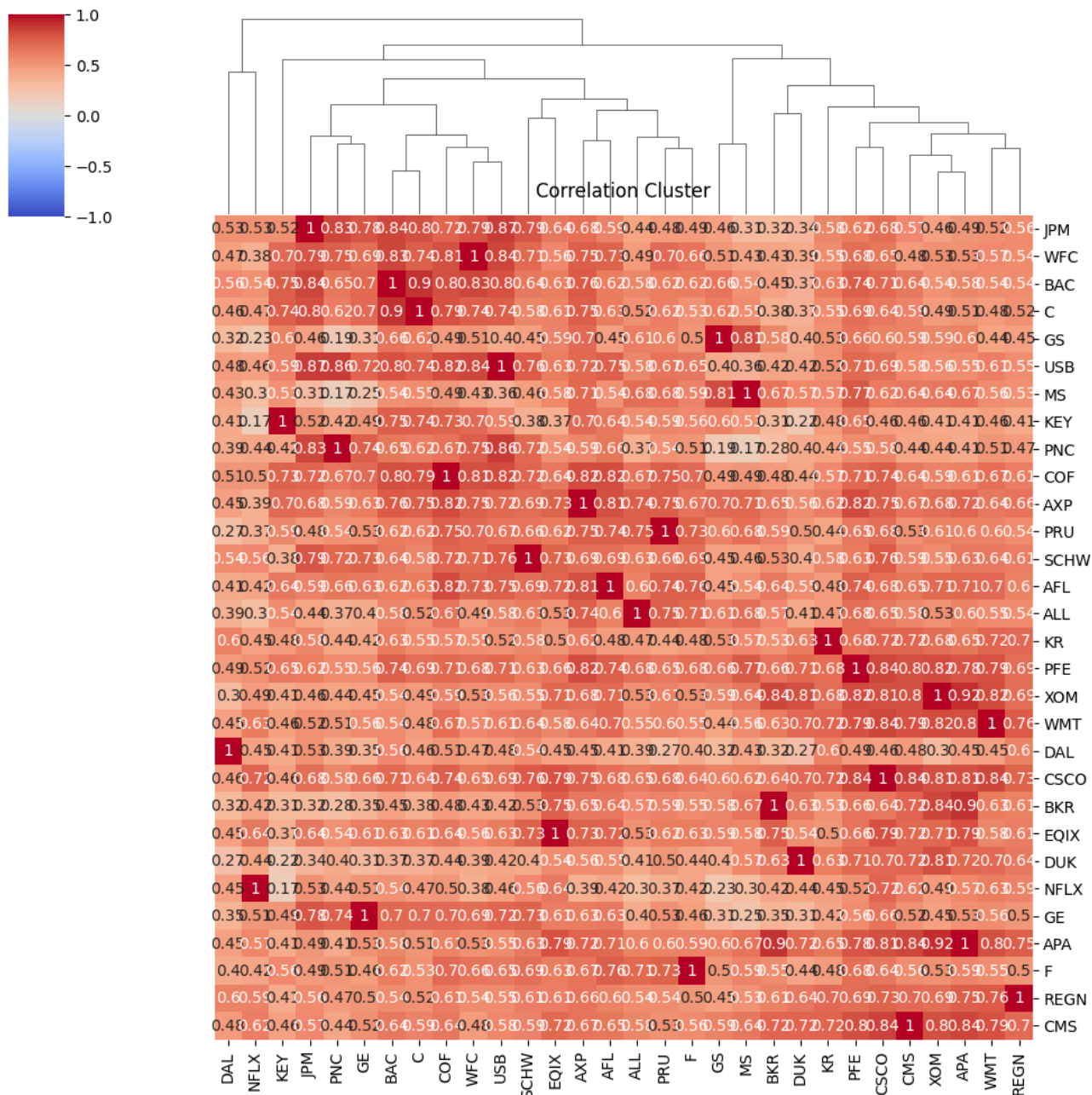


```
plt.figure(figsize=(16,6))
mask = np.triu(np.ones_like(ret_cor))
heatmap =sns.heatmap(ret_cor, mask=mask, vmin=-1, vmax=1, cmap="coolwarm", annot=True)
heatmap.set_title("Correlation Heatmap", fontdict={"fontsize" : 12}, pad = 12);
```



```
plt.figure(figsize=(8,3))
mask = np.triu(np.ones_like(ret_cor))
clustermap = sns.clustermap(ret_cor, row_cluster=False, vmin=-1, vmax=1, cmap="coolwarm",
# Setting the title for the clustermap
clustermap.ax_heatmap.set_title("Correlation Cluster", fontdict={"fontsize": 12}, pad=12)

plt.show()
```


 <Figure size 800x300 with 0 Axes>


✓ Analyzing Risk and Return Profiles of Clustered Securities

In this task, we aim to analyze the risk and return profiles of securities grouped into clusters. We start by clustering the securities based on correlation of their daily historical returns using hierarchical clustering. The clustering process groups securities that exhibit similar return correlation patterns together, providing insights into their relationships and potential similarities in market behavior.

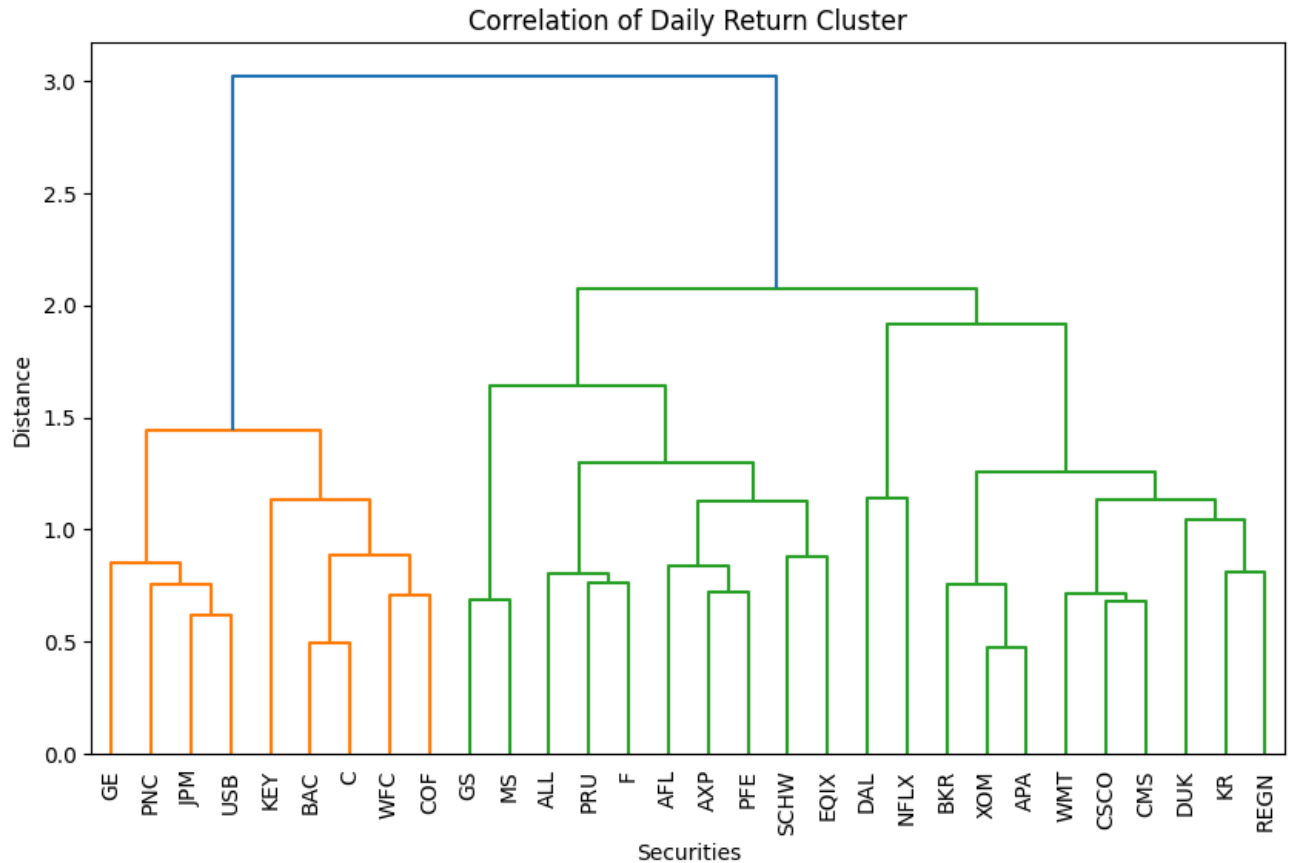
- 1. Clustering:** We perform hierarchical clustering on the correlation of historical returns of the securities. This clustering method groups securities into clusters based on their similarities in return trends and behavior. Each cluster represents a group of securities that tend to move together in the market.
- 2. Risk and Return Analysis:** For each cluster identified, we conduct a comprehensive analysis of the risk and return profiles of the securities within each cluster. This analysis includes calculating metrics such as average return, volatility, and Sharpe ratio for each security. These metrics help us understand the risk-return trade-offs associated with each cluster, providing valuable insights into the performance and risk characteristics of the securities.
- 3. Interpretation:** Finally, we interpret the results of the analysis, drawing conclusions about the risk and return profiles of the clustered securities. We analyze the average returns, volatility, and Sharpe ratio for each cluster, comparing them to understand the differences in performance and risk-adjusted returns. Additionally, we examine the average pairwise correlation within each cluster to understand the degree of correlation among securities within the same cluster.

```
# Calculate pairwise distances
pairwise_distances = np.sqrt(1 - ret_cor)

# Perform hierarchical clustering
linkage = hierarchy.linkage(pairwise_distances, method='ward')

# Determine the optimal number of clusters using a dendrogram
plt.figure(figsize=(10, 6))
dn = hierarchy.dendrogram(linkage, labels=tickers, leaf_rotation=90)
plt.title('Correlation of Daily Return Cluster')
plt.ylabel('Distance')
plt.xlabel('Securities')
plt.show()
```

```
<ipython-input-8-bb10b3e08a04>:5: ClusterWarning: scipy.cluster: The symmetric non-ne
linkage = hierarchy.linkage(pairwise_distances, method='ward')
```



```
# Assuming 'optimal_num_clusters' is the optimal number of clusters determined from the d
# And 'original_data' is your original dataset containing the securities
```

```
clusters = hierarchy.fcluster(linkage, 2, criterion='maxclust')
```

```
# Create a dictionary to store securities in each cluster
```

```
clustered_securities = {}
for i, security in enumerate(df.columns):
    cluster = clusters[i]
    if cluster not in clustered_securities:
        clustered_securities[cluster] = []
    clustered_securities[cluster].append(security)
```

```
# Print the securities in each cluster
```

```
for cluster, securities in clustered_securities.items():
    print(f'Cluster {cluster}: {"", ".join(securities)}')
```

```
Cluster 1: JPM, WFC, BAC, C, USB, KEY, PNC, COF, GE
Cluster 2: GS, MS, AXP, PRU, SCHW, AFL, ALL, KR, PFE, XOM, WMT, DAL, CSCO, BKR, EQIX,
```

```
# Define the clusters as provided
clusters = {
    1: ['JPM', 'WFC', 'BAC', 'C', 'USB', 'KEY', 'PNC', 'COF', 'GE'],
    2: ['GS', 'MS', 'AXP', 'PRU', 'SCHW', 'AFL', 'ALL', 'KR', 'PFE', 'XOM', 'WMT', 'DAL'],
}

# Print the dictionary
print(clusters)
```

```
➦ {1: ['JPM', 'WFC', 'BAC', 'C', 'USB', 'KEY', 'PNC', 'COF', 'GE'], 2: ['GS', 'MS', 'AX
```

```
# Risk and Return Analysis
cluster_metrics = {}
for cluster, securities in clusters.items():
    cluster_returns = df[securities]
    avg_return = cluster_returns.mean()
    volatility = cluster_returns.std()
    excess_returns = cluster_returns.mean() # Assuming risk-free rate is zero for simpli
    sharpe_ratio = excess_returns / volatility

    cluster_metrics[cluster] = {
        'Average Return': avg_return,
        'Volatility': volatility,
        'Sharpe Ratio': sharpe_ratio
    }
```

```
# Correlation Analysis
cluster_correlations = {}
for cluster, securities in clusters.items():
    cluster_returns = df[securities]
    pairwise_correlations = cluster_returns.corr()
    avg_correlation = pairwise_correlations.mean().mean()

    cluster_correlations[cluster] = avg_correlation
```

```
# Print or visualize the results
for cluster, metrics in cluster_metrics.items():
    print(f'Cluster {cluster}:')
    print(pd.DataFrame(metrics))

for cluster, avg_correlation in cluster_correlations.items():
    print(f'Average pairwise correlation in Cluster {cluster}: {avg_correlation}')
```

```
➦ Cluster 1:
      Average Return  Volatility  Sharpe Ratio
JPM          0.002304    0.078245    0.029442
WFC          0.002759    0.068376    0.040345
```

BAC	-0.003077	0.099449	-0.030940
C	-0.004175	0.096793	-0.043135
USB	-0.001354	0.048241	-0.028069
KEY	0.008212	0.136987	0.059950
PNC	-0.001294	0.057035	-0.022691
COF	-0.000850	0.079223	-0.010730
GE	-0.007468	0.052666	-0.141804

Cluster 2:

	Average Return	Volatility	Sharpe Ratio
GS	-0.011093	0.078768	-0.140837
MS	-0.009401	0.177123	-0.053078
AXP	-0.007992	0.070095	-0.114020
PRU	-0.018070	0.104853	-0.172337
SCHW	-0.003709	0.073896	-0.050187
AFL	-0.004483	0.068753	-0.065210
ALL	-0.012234	0.063787	-0.191789
KR	0.000149	0.029944	0.004985
PFE	-0.000961	0.038600	-0.024893
XOM	0.001122	0.061580	0.018221
WMT	-0.001445	0.035174	-0.041091
DAL	0.004547	0.086603	0.052506
CSCO	-0.005642	0.050228	-0.112334
BKR	-0.016245	0.077633	-0.209249
EQIX	-0.004616	0.055157	-0.083680
DUK	0.000187	0.043244	0.004313
NFLX	-0.005761	0.053486	-0.107705
APA	-0.004196	0.076324	-0.054976
F	-0.012552	0.085065	-0.147558
REGN	-0.001337	0.058693	-0.022776
CMS	-0.005097	0.044846	-0.113651

Average pairwise correlation in Cluster 1: 0.7642305128998724

Average pairwise correlation in Cluster 2: 0.631251227750594

Result: Cluster 1 appears to consist of securities with higher average returns, lower volatility, and higher correlation, suggesting they may represent a group of financially robust companies or belong to similar industries. In contrast, Cluster 2 comprises securities with lower average returns, higher volatility, and slightly lower correlation, indicating they may represent a more diverse set of companies or industries with varying performance levels.

✓ Extracting the data set of each Cluster

```
tickers_1 = ['JPM', 'WFC', 'BAC', 'C', 'USB', 'KEY', 'PNC', 'COF', 'GE']

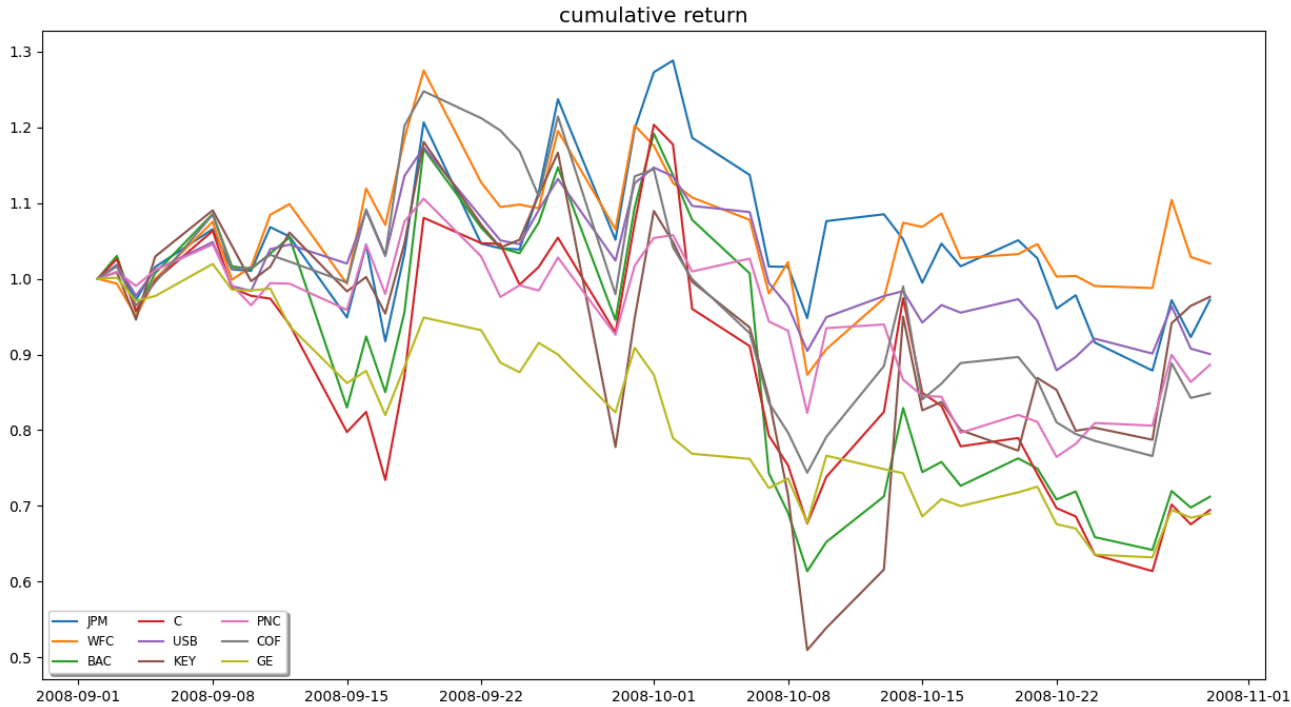
len(tickers_1)

start_date = "2008-09-01"
end_date = "2008-10-31"

df_1=pd.DataFrame()
df_ret_1=pd.DataFrame()
for tick in tickers_1:
    price_1=yf.download(tick, start=start_date, end=end_date)
    plt.plot(price_1["Adj Close"] / price_1["Adj Close"][0], label=tick)
    price_1=price_1.rename(columns={"Adj Close" : tick})
    price_1[tick+"ret"]=price_1[tick].pct_change()
    df_1=pd.concat([df_1, price_1[[tick]]], axis=1)
    df_ret_1=pd.concat([df_1, price_1[[tick+"ret"]]], axis=1)
pdata_1 = df_1.to_numpy()
pdata_dates_1 = pd.to_datetime(price_1.index, format = "%Y-%m-%d")
legend = plt.legend(loc="lower left", shadow=True, fontsize="small", ncol=3)
#legend_title = plt.gca().add_artist(plt.Text(0, 0, "Stock Tickers", fontsize="small", fo
title="cumulative return"
plt.title(title, fontsize="x-large")
fig=plt.gcf()
fig.set_size_inches(15,8)
plt.show()
```



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

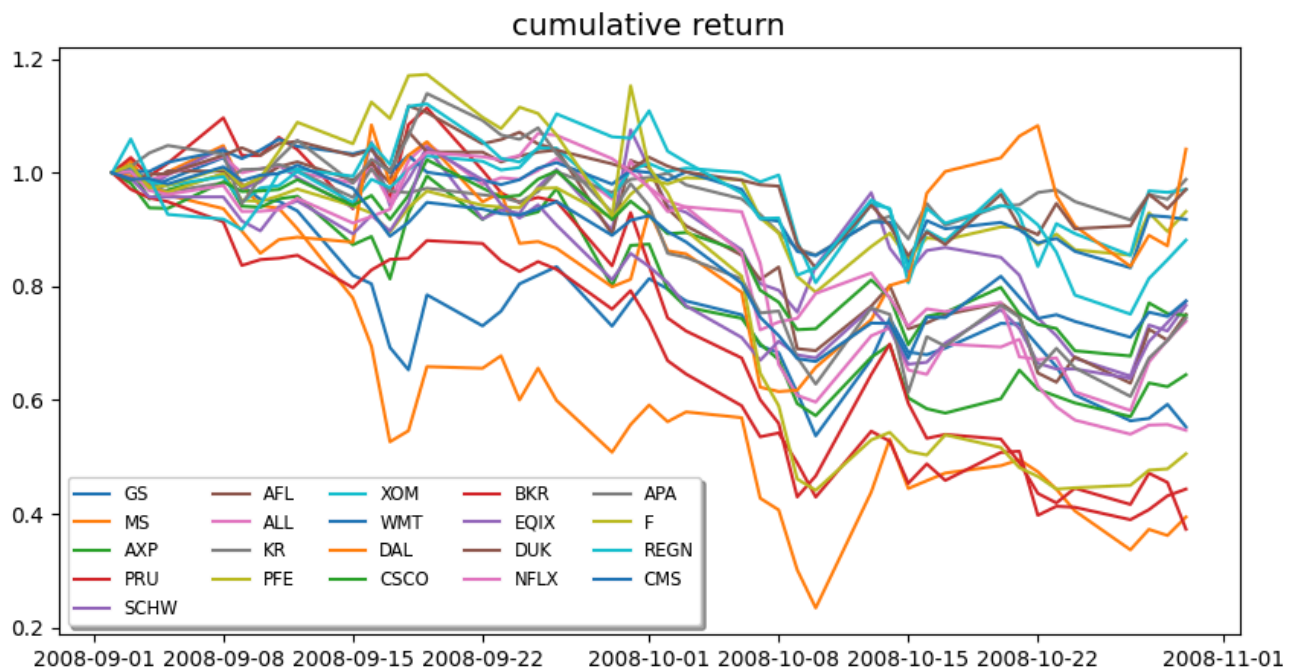


```
tickers_2 = ['GS', 'MS', 'AXP', 'PRU', 'SCHW', 'AFL', 'ALL', 'KR', 'PFE', 'XOM', 'WMT', ' '

len(tickers_2)

start_date = "2008-09-01"
end_date = "2008-10-31"

df_2=pd.DataFrame()
df_ret_2=pd.DataFrame()
for tick in tickers_2:
    price_2=yf.download(tick, start=start_date, end=end_date)
    plt.plot(price_2["Adj Close"] / price_2["Adj Close"][0], label=tick)
    price_2=price_2.rename(columns={"Adj Close" : tick})
    price_2[tick+"ret"]=price_2[tick].pct_change()
    df_2=pd.concat([df_2, price_2[[tick]]], axis=1)
    df_ret_2=pd.concat([df_2, price_2[[tick+"ret"]]], axis=1)
pdata_2 = df_2.to_numpy()
pdata_dates_2 = pd.to_datetime(price_2.index, format = "%Y-%m-%d")
legend = plt.legend(loc="lower left", shadow=True, fontsize="small", ncol=5)
#legend_title = plt.gca().add_artist(plt.Text(0, 0, "Stock Tickers", fontsize="small", fo
title="cumulative return"
plt.title(title, fontsize="x-large")
fig=plt.gcf()
fig.set_size_inches(10,5)
plt.show()
```


[illegible]

- ✧ K - Bandit Algorithm

In this task, we're tackling the K-bandit problem for stock selection, where our aim is to effectively allocate resources among a set of 30 stocks, comprising 15 financial and 15 non-financial entities. We understand the challenge ahead: dynamically choosing the most promising stocks over time. Each stock serves as a "bandit arm" in this problem, and our task involves

continuously evaluating and updating our selection strategy based on both historical performance and prevailing market conditions. To address this, we're planning to employ K-bandit algorithms, such as Upper-Confidence Bound (UCB) or epsilon-greedy methods. By doing so, we'll systematically explore and exploit the potential of different stocks, ensuring a balanced approach between exploration and exploitation to optimize our portfolio's performance.\

Firstly, we will define the two functions which are the core steps of the algorithm:

- an optimal action choice given information about the past
- an updating of the expected reward from choosing the given action.

✓ Defining the functions that govern the bandit problem

Define the functions that govern the bandit problem

```
def optimal_action(qvalue, eps): # noQA E203
    """
    Determines what is the action to take given a measure of past
    expected rewards across actions. With probability eps the action
    is not the greedy one
    """
    nactions = qvalue.shape[0]
    action_hat = np.where(qvalue == np.max(qvalue))

    if rand() <= eps:
        randnum = rand()
        for aa in range(nactions):
            if randnum < (aa + 1) / nactions: # noQA E203
                break
    elif action_hat[0].shape[0] > 1: # noQA E203
        # Randomize action when ties
        randnum = rand()
        for aa in range(action_hat[0].shape[0]): # noQA E203
            if randnum < (aa + 1) / action_hat[0].shape[0]: # noQA E203
                break
        aa = action_hat[0][aa]
    else:
        aa = np.argmax(qvalue)

    return aa

def reward_update(action, reward, qvalue_old, alpha): # noQA E203
    qvalue_new = qvalue_old.copy()

    qvalue_new[action] = qvalue_old[action] + alpha * (reward - qvalue_old[action])

    return qvalue_new
```

› UCB WEIGHTED ALGORITHM

[] ↳ 18 cells hidden

› EPSILON - GREEDY ALGORITHM

[] ↳ 18 cells hidden

✓ **updating 30 data series.**

For this task, we are going to update the data of each of the selected securities by downloading the data for a different period of time say March 2020 to April 2020 perhaps to investigate the impact of the beginning of covid-19.

For performing the algorithms, we would keep the holding period but vary the exploration-exploitation parameter ϵ and the updating parameter α .

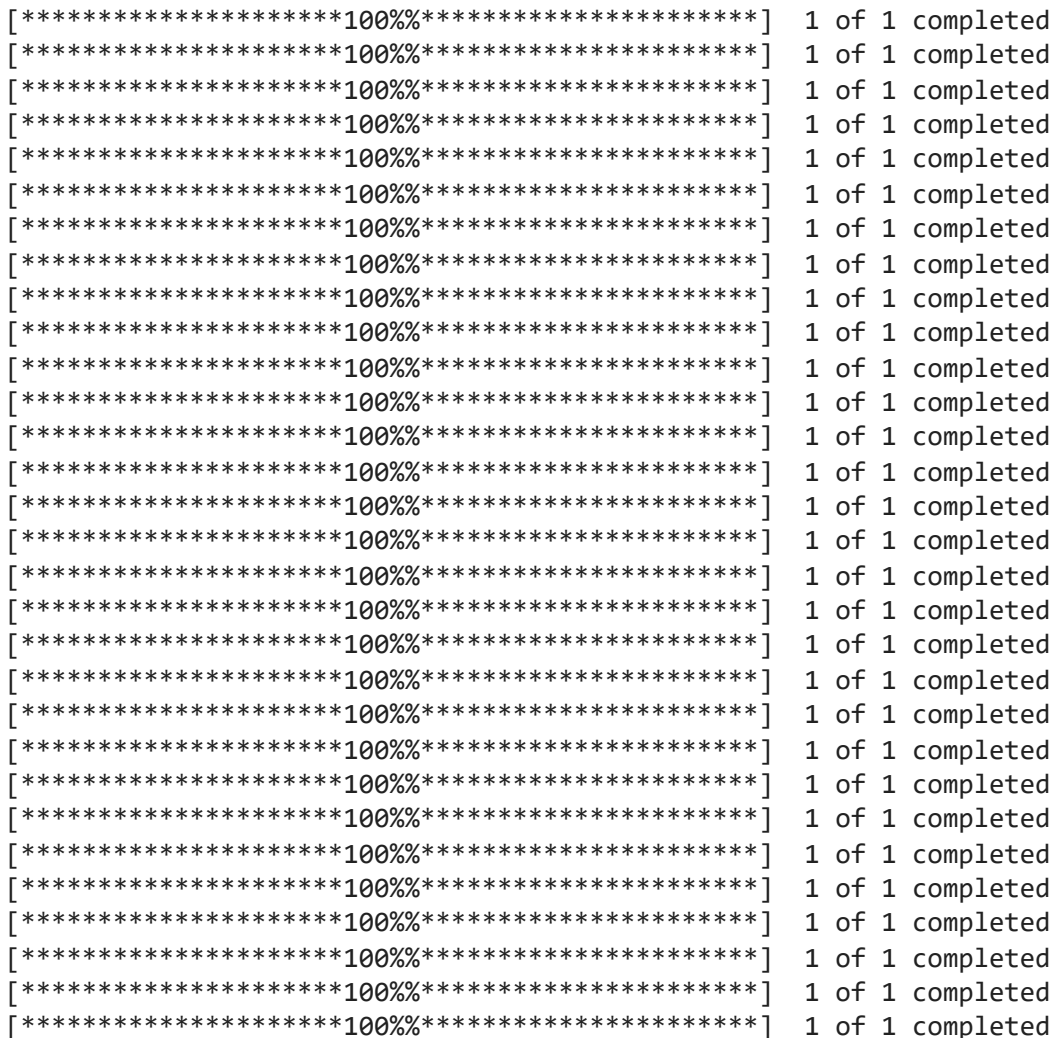
We are going to perform the algorithm on the 30 stocks first and then on the selected portfolios namely cluster 1 and 2. Using a holding period of 1 day, we are going to fix an ϵ -greedy policy that uses $\epsilon = 0.15$. Then, we are going to compare the performance of the updating policy that averages across the past history of rewards and the updating policy that uses a fixed parameter $\alpha = 0.5$. We will also adjust the degree of "unexploration" of the action relative to the steps completed with the introduction of UBC_WEIGHT= 2.

```
tickers = ["JPM", "WFC", "BAC", "C", "GS", "USB", "MS", "KEY", "PNC", "COF", "AXP",
           "PRU", "SCHW", "AFL", "ALL", "KR", "PFE", "XOM", "WMT", "DAL", "CSCO", "BKR",
           "EQIX", "DUK", "NFLX", "GE", "APA", "F", "REGN", "CMS"
          ]

len(tickers)

start_date = "2020-03-01"
end_date = "2020-04-30"

df=pd.DataFrame()
df_ret=pd.DataFrame()
for tick in tickers:
    price=yf.download(tick, start=start_date, end=end_date)
    plt.plot(price["Adj Close"] / price["Adj Close"][0], label=tick)
    price=price.rename(columns={"Adj Close" : tick})
    price[tick+"ret"]=price[tick].pct_change()
    df=pd.concat([df, price[[tick]]], axis=1)
    df_ret=pd.concat([df, price[[tick+"ret"]]], axis=1)
pdata = df.to_numpy()
pdata_dates = pd.to_datetime(price.index, format = "%Y-%m-%d")
legend = plt.legend(loc="lower left", shadow=True, fontsize="small", ncol=5)
#legend_title = plt.gca().add_artist(plt.Text(0, 0, "Stock Tickers", fontsize="small", fo
title="cumulative return"
plt.title(title, fontsize="x-large")
fig=plt.gcf()
fig.set_size_inches(15,8)
plt.show()
```



The chart displays the performance of 25 US stock indices from March 1, 2020, to May 1, 2020. The indices are: JPM, WFC, BAC, C, GS, USB, MS, KEY, PNC, COF, AXP, PRU, SCHW, AFL, ALL, KR, XOM, DAL, CSCO, BKR, EQIX, DUK, WMT, NFLX, GE, APA, F, and REGN. The chart shows a general upward trend for most indices, with a notable dip in early April followed by a recovery. The indices are color-coded and labeled in the legend.

```
tickers_1 = ['JPM', 'WFC', 'BAC', 'C', 'USB', 'KEY', 'PNC', 'COF', 'GE']

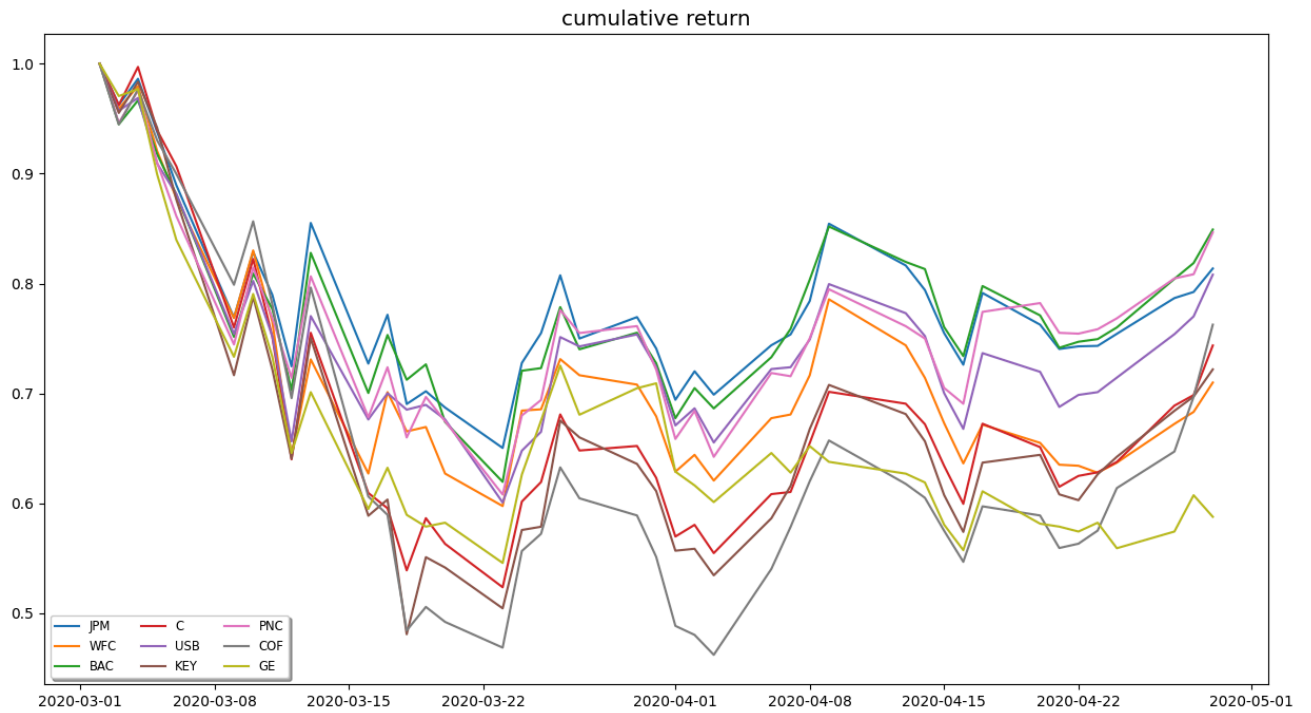
len(tickers_1)

start_date = "2020-03-01"
end_date = "2020-04-30"

df_1=pd.DataFrame()
df_ret_1=pd.DataFrame()
for tick in tickers_1:
    price_1=yf.download(tick, start=start_date, end=end_date)
    plt.plot(price_1["Adj Close"] / price_1["Adj Close"][0], label=tick)
    price_1=price_1.rename(columns={"Adj Close" : tick})
    price_1[tick+"ret"]=price_1[tick].pct_change()
    df_1=pd.concat([df_1, price_1[[tick]]], axis=1)
    df_ret_1=pd.concat([df_1, price_1[[tick+"ret"]]], axis=1)
pdata_1 = df_1.to_numpy()
pdata_dates_1 = pd.to_datetime(price_1.index, format = "%Y-%m-%d")
legend = plt.legend(loc="lower left", shadow=True, fontsize="small", ncol=3)
#legend_title = plt.gca().add_artist(plt.Text(0, 0, "Stock Tickers", fontsize="small", fo
title="cumulative return"
plt.title(title, fontsize="x-large")
fig=plt.gcf()
fig.set_size_inches(15,8)
plt.show()
```



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



```
tickers_2 = ['GS', 'MS', 'AXP', 'PRU', 'SCHW', 'AFL', 'ALL', 'KR', 'PFE', 'XOM', 'WMT', ' '

len(tickers_2)

start_date = "2020-03-01"
end_date = "2020-04-30"

df_2=pd.DataFrame()
df_ret_2=pd.DataFrame()
for tick in tickers_2:
    price_2=yf.download(tick, start=start_date, end=end_date)
    plt.plot(price_2["Adj Close"] / price_2["Adj Close"][0], label=tick)
    price_2=price_2.rename(columns={"Adj Close" : tick})
    price_2[tick+"ret"]=price_2[tick].pct_change()
    df_2=pd.concat([df_2, price_2[[tick]]], axis=1)
    df_ret_2=pd.concat([df_2, price_2[[tick+"ret"]]], axis=1)
pdata_2 = df_2.to_numpy()
pdata_dates_2 = pd.to_datetime(price_2.index, format = "%Y-%m-%d")
legend = plt.legend(loc="lower left", shadow=True, fontsize="small", ncol=1)
#legend_title = plt.gca().add_artist(plt.Text(0, 0, "Stock Tickers", fontsize="small", fo
title="cumulative return"
plt.title(title, fontsize="x-large")
fig=plt.gcf()
fig.set_size_inches(14,8)
plt.show()
```


[illegible]

✓ Performing the Algorithms on the new data sets.

✓ 1. UCB ALgorithm on the 30 Securities

```
# Bandit problem for stock selection

NK = pdata.shape[1]
EPSILON = 0.15
ALPHA = 0.5
NEPISODES = 1000
HOLD = 1
TMAX = pdata.shape[0] - HOLD

# New Parameter
UBC_WEIGHT= 2 # degree of exploration parameter

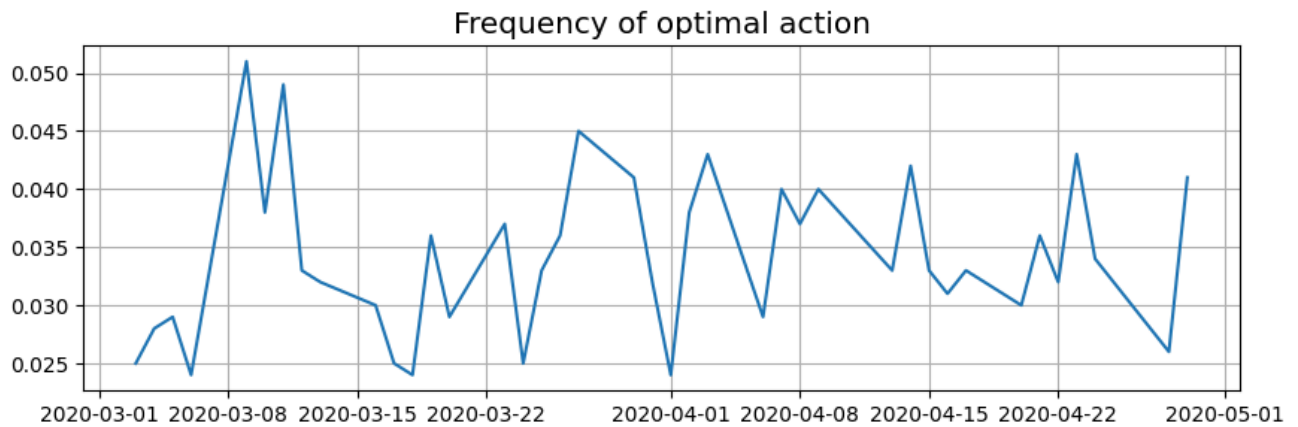
seed(1234)

reward_avg = np.zeros((TMAX))
optimal_avg = np.zeros((TMAX))

reward_queue = np.zeros((HOLD, 2))

for run in range(NEPISODES): # noQA E203
    # Initialize q function and actions record
    qvalue = np.zeros((NK))
    qvalue_up = np.zeros((NK))
    nactions = np.zeros((NK))
    for tt in range(TMAX): # noQA E203
        aa_opt = optimal_action(qvalue_up, EPSILON)
        nactions[aa_opt] += 1
        # Compute reward as return over holding period
        reward_queue[HOLD - 1, 0] = (pdata[tt + HOLD, aa_opt] - pdata[tt, aa_opt]) / pdat
        reward_queue[HOLD - 1, 1] = aa_opt
        # Update Q function using action chosen HOLD days before
        qvalue = reward_update(int(reward_queue[0, 1]), reward_queue[0, 0], qvalue, ALPHA)
        #qvalue = reward_update(int(reward_queue[0, 1]), reward_queue[0, 0], qvalue, 1/na)
        #print(qvalue)
        # Upper-confidence adjustment
        qvalue_up = np.zeros((NK))
        for aa in range(NK):
            # If an action has not been visited, simply give it maximum value across acti
            if nactions[aa] == 0: # noQA E203
                qvalue_up[aa] = np.max(qvalue) + 1.0
            else:
                qvalue_up[aa] = qvalue[aa] + UBC_WEIGHT * np.sqrt(np.log(tt + 1) / nactio
        reward_queue[0 : HOLD - 1, :] = reward_queue[1:HOLD, :] # noQA E203
        reward_avg[tt] += reward_queue[HOLD - 1, 0] / NEPISODES
        optimal_avg[tt] += (aa_opt == np.argmax((pdata[tt + HOLD, :] - pdata[tt, :])) / pd
```

```
plt.plot(pdata_dates[HOLD : pdata.shape[0]], optimal_avg) # noQA E203
plt.title("Frequency of optimal action", fontsize="x-large")
plt.grid(True)
fig = plt.gcf()
fig.set_size_inches(10, 3)
plt.show()
```

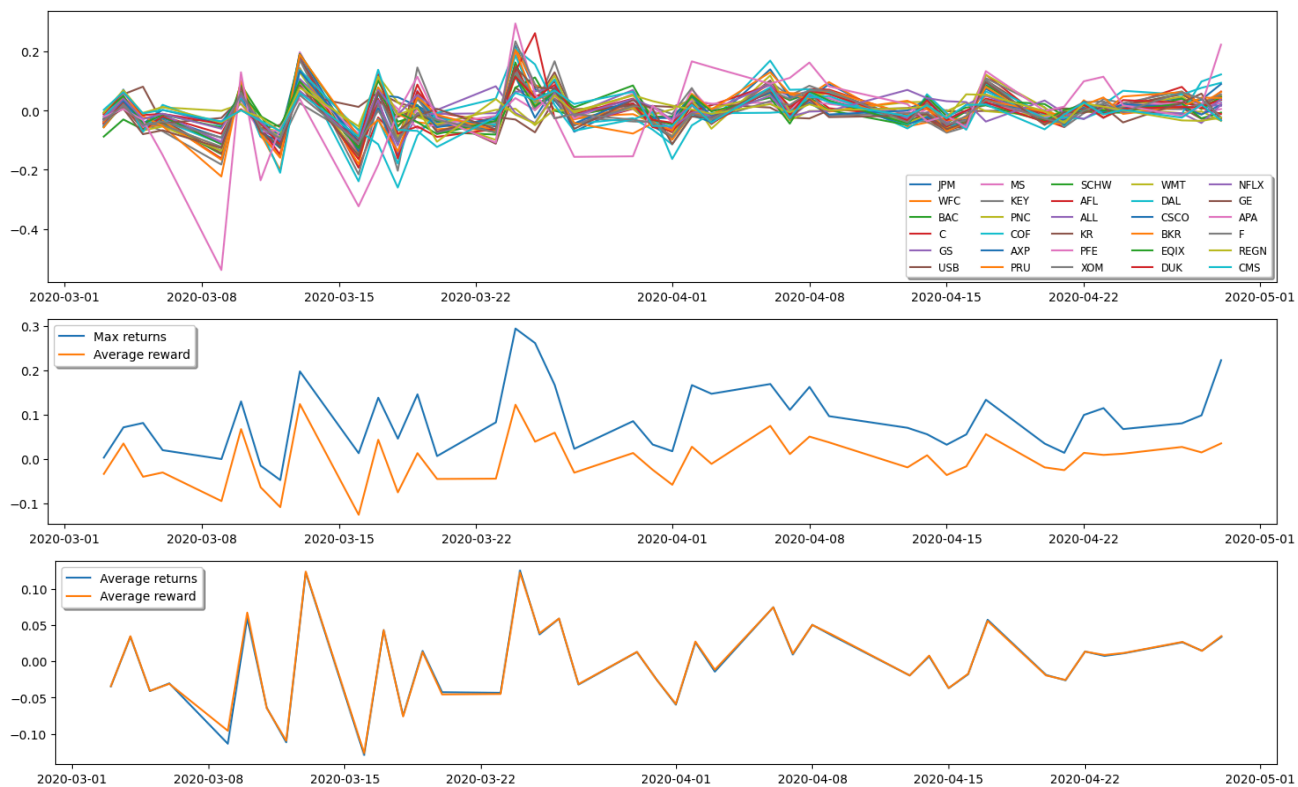


```
plt.plot(pdata_dates[HOLD : pdata.shape[0]],(pdata[HOLD : pdata.shape[0], :] - pdata[0:TM
#plt.legend(tickers)
legend = plt.legend(tickers, loc="lower right", shadow=True, fontsize="small", ncol=5)
fig = plt.gcf()
fig.set_size_inches(18, 4)
plt.show()
```

```
plt.plot(pdata_dates[HOLD : pdata.shape[0]],np.max((pdata[HOLD : pdata.shape[0], :] - pda
plt.plot(pdata_dates[HOLD : pdata.shape[0]], reward_avg, label="Average reward") # noQA
legend = plt.legend(loc="upper left", shadow=True)
fig = plt.gcf()
fig.set_size_inches(18, 3)
plt.show()
```

```
plt.plot(pdata_dates[HOLD : pdata.shape[0]],np.mean((pdata[HOLD : pdata.shape[0], :] - pd
plt.plot(pdata_dates[HOLD : pdata.shape[0]], reward_avg, label="Average reward") # noQA
legend = plt.legend(loc="upper left", shadow=True)
fig = plt.gcf()
fig.set_size_inches(18, 3)
plt.show()
```

```
# Average frequency of optimal action
print(np.mean(optimal_avg))
# Average annualized return from holding the equally-weighted portfolio
print((1+ np.mean((pdata[HOLD : pdata.shape[0], :] - pdata[0:TMAX, :]) / pdata[0:TMAX, :])
# Average annualized return from holding the Bandit portfolio
print((1 + np.mean(reward_avg)) ** (250 / HOLD) - 1,np.sqrt(250 / HOLD) * np.std(reward_a
```



0.03402439024390246

-0.29939971197939674 0.8764664967605831

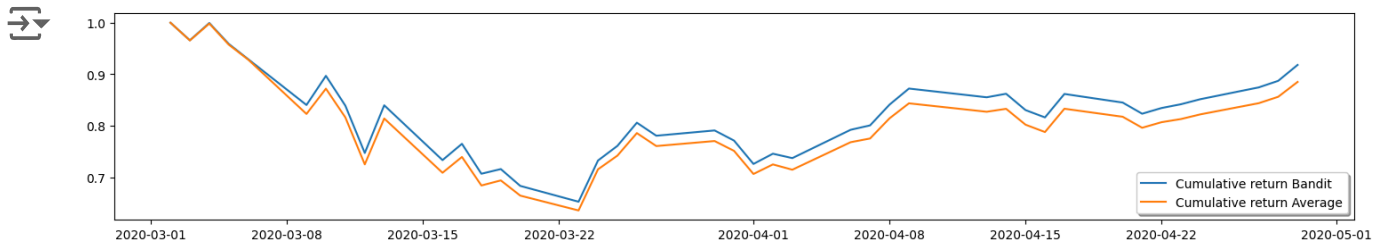
-0.13558309555866122 0.8635727401935961

```

return_cumulative = np.zeros((TMAX + 1, 2))
return_cumulative[0, 0] = 1
return_cumulative[0, 1] = 1
for tt in range(1, TMAX + 1): # noQA E203
    return_cumulative[tt, 0] = return_cumulative[tt - 1, 0] * (1 + reward_avg[tt - 1])
    rmean = np.mean((pdata[tt + HOLD - 1, :] - pdata[tt - 1, :]) / pdata[tt - 1, :]) # n
    return_cumulative[tt, 1] = return_cumulative[tt - 1, 1] * (1 + rmean) # noQA E203

plt.plot(pdata_dates[HOLD - 1 : pdata.shape[0]], return_cumulative[:, 0], label="Cumulative")
plt.plot(pdata_dates[HOLD - 1 : pdata.shape[0]], return_cumulative[:, 1], label="Cumulative")
legend = plt.legend(loc="lower right", shadow=True)
fig = plt.gcf()
fig.set_size_inches(18, 3)
plt.show()

```



✓ UCB Algorithm on Updated Cluster 1 securities

```

# Bandit problem for stock selection

NK = pdata_1.shape[1]
EPSILON = 0.15
ALPHA = 0.5
NEPISODES = 1000
HOLD = 1
TMAX = pdata_1.shape[0] - HOLD

# New Parameter
UBC_WEIGHT= 2 # degree of exploration parameter

seed(1234)

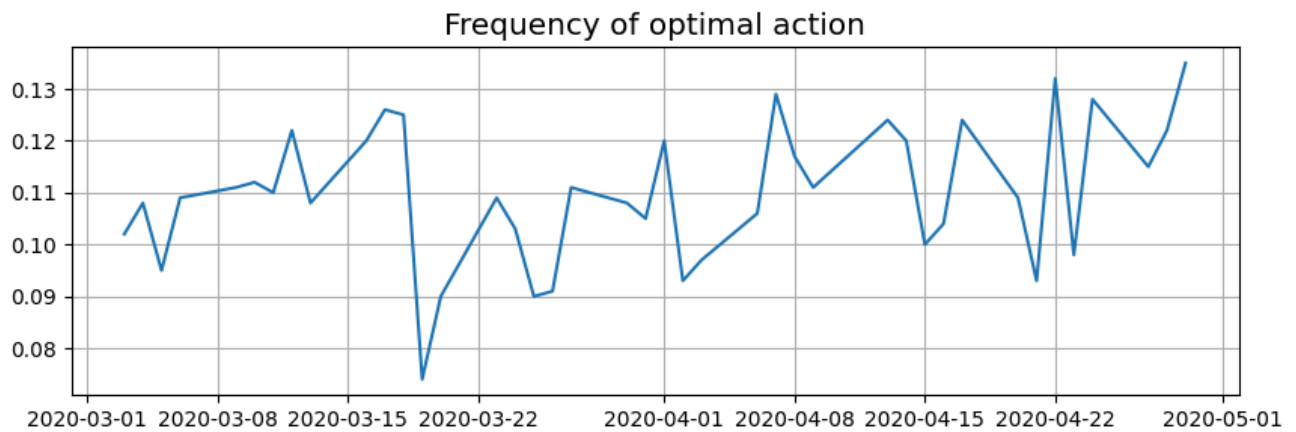
reward_avg = np.zeros((TMAX))
optimal_avg = np.zeros((TMAX))

reward_queue = np.zeros((HOLD, 2))

for run in range(NEPISODES): # noQA E203
    # Initialize q function and actions record
    qvalue = np.zeros((NK))
    qvalue_up = np.zeros((NK))
    nactions = np.zeros((NK))
    for tt in range(TMAX): # noQA E203
        aa_opt = optimal_action(qvalue_up, EPSILON)
        nactions[aa_opt] += 1
        # Compute reward as return over holding period
        reward_queue[HOLD - 1, 0] = (pdata_1[tt + HOLD, aa_opt] - pdata_1[tt, aa_opt]) /
        reward_queue[HOLD - 1, 1] = aa_opt
        # Update Q function using action chosen HOLD days before
        qvalue = reward_update(int(reward_queue[0, 1]), reward_queue[0, 0], qvalue, ALPHA)
        #qvalue = reward_update(int(reward_queue[0, 1]), reward_queue[0, 0], qvalue, 1/na)
        #print(qvalue)
        # Upper-confidence adjustment
        qvalue_up = np.zeros((NK))
        for aa in range(NK):
            # If an action has not been visited, simply give it maximum value across acti
            if nactions[aa] == 0: # noQA E203
                qvalue_up[aa] = np.max(qvalue) + 1.0
            else:
                qvalue_up[aa] = qvalue[aa] + UBC_WEIGHT * np.sqrt(np.log(tt + 1) / nactio
        reward_queue[0 : HOLD - 1, :] = reward_queue[1:HOLD, :] # noQA E203
        reward_avg[tt] += reward_queue[HOLD - 1, 0] / NEPISODES
        optimal_avg[tt] += (aa_opt == np.argmax((pdata_1[tt + HOLD, :] - pdata_1[tt, :]))

plt.plot(pdata_dates_1[HOLD : pdata_1.shape[0]], optimal_avg) # noQA E203
plt.title("Frequency of optimal action", fontsize="x-large")
plt.grid(True)
fig = plt.gcf()
fig.set_size_inches(10, 3)
plt.show()

```

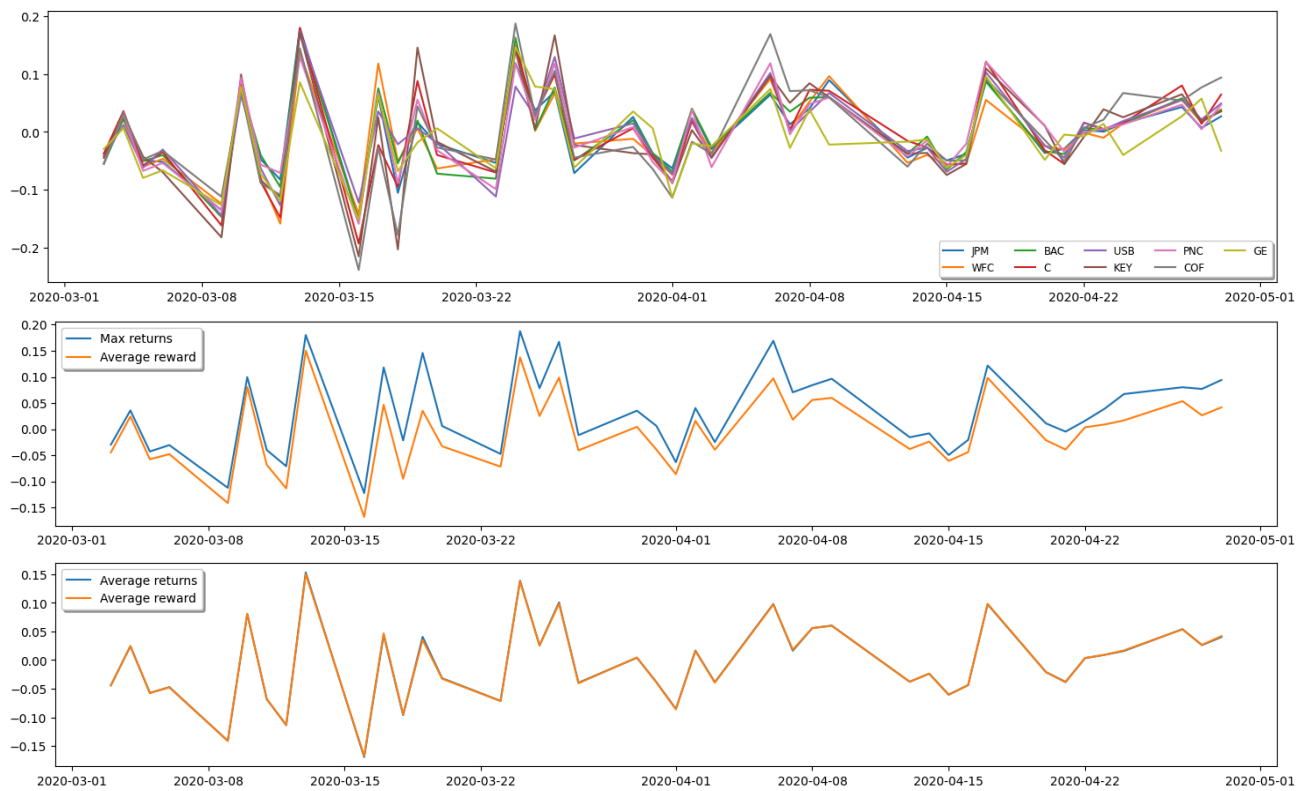


```
plt.plot(pdata_dates_1[HOLD : pdata_1.shape[0]],(pdata_1[HOLD : pdata_1.shape[0], :] - pd
#plt.legend(tickers)
legend = plt.legend(tickers_1, loc="lower right", shadow=True, fontsize="small", ncol=5)
fig = plt.gcf()
fig.set_size_inches(18, 4)
plt.show()
```

```
plt.plot(pdata_dates_1[HOLD : pdata_1.shape[0]],np.max((pdata_1[HOLD : pdata_1.shape[0],
plt.plot(pdata_dates_1[HOLD : pdata_1.shape[0]], reward_avg, label="Average reward") # n
legend = plt.legend(loc="upper left", shadow=True)
fig = plt.gcf()
fig.set_size_inches(18, 3)
plt.show()
```

```
plt.plot(pdata_dates_1[HOLD : pdata_1.shape[0]],np.mean((pdata_1[HOLD : pdata_1.shape[0],
plt.plot(pdata_dates_1[HOLD : pdata_1.shape[0]], reward_avg, label="Average reward") # n
legend = plt.legend(loc="upper left", shadow=True)
fig = plt.gcf()
fig.set_size_inches(18, 3)
plt.show()
```

```
# Average frequency of optimal action
print(np.mean(optimal_avg))
# Average annualized return from holding the equally-weighted portfolio
print((1+ np.mean((pdata_1[HOLD : pdata_1.shape[0], :] - pdata_1[0:TMAX, :]) / pdata_1[0:
# Average annualized return from holding the Bandit portfolio
print((1 + np.mean(reward_avg)) ** (250 / HOLD) - 1,np.sqrt(250 / HOLD) * np.std(reward_a
```



0.10990243902439029

-0.6299785473623191 1.1201008812707098

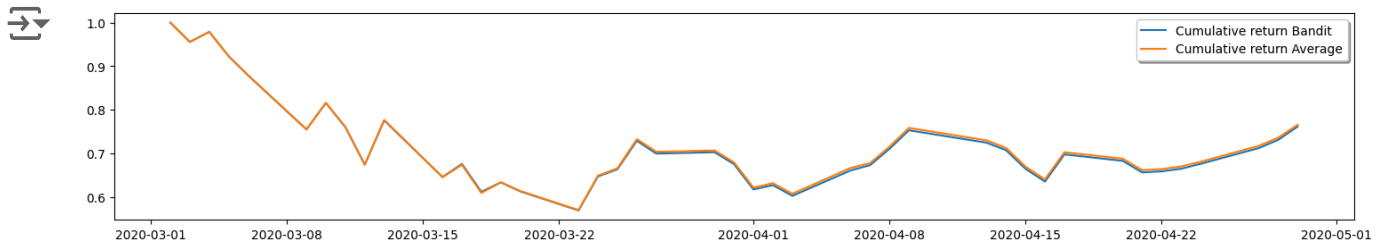
-0.643948625587246 1.1143101629460916


```

return_cumulative = np.zeros((TMAX + 1, 2))
return_cumulative[0, 0] = 1
return_cumulative[0, 1] = 1
for tt in range(1, TMAX + 1): # noQA E203
    return_cumulative[tt, 0] = return_cumulative[tt - 1, 0] * (1 + reward_avg[tt - 1])
    rmean = np.mean((pdata_1[tt + HOLD - 1, :] - pdata_1[tt - 1, :]) / pdata_1[tt - 1, :])
    return_cumulative[tt, 1] = return_cumulative[tt - 1, 1] * (1 + rmean) # noQA E203

plt.plot(pdata_dates_1[HOLD - 1 : pdata_1.shape[0]], return_cumulative[:, 0], label="Cumula
plt.plot(pdata_dates_1[HOLD - 1 : pdata_1.shape[0]], return_cumulative[:, 1], label="Cumula
legend = plt.legend(loc="upper right", shadow=True)
fig = plt.gcf()
fig.set_size_inches(18, 3)
plt.show()

```



✓ UCB Algorithm on Updated Cluster 2 securities

```

# Bandit problem for stock selection

NK = pdata_2.shape[1]
EPSILON = 0.15
ALPHA = 0.5
NEPISODES = 1000
HOLD = 1
TMAX = pdata_2.shape[0] - HOLD

# New Parameter
UBC_WEIGHT= 2 # degree of exploration parameter

seed(1234)

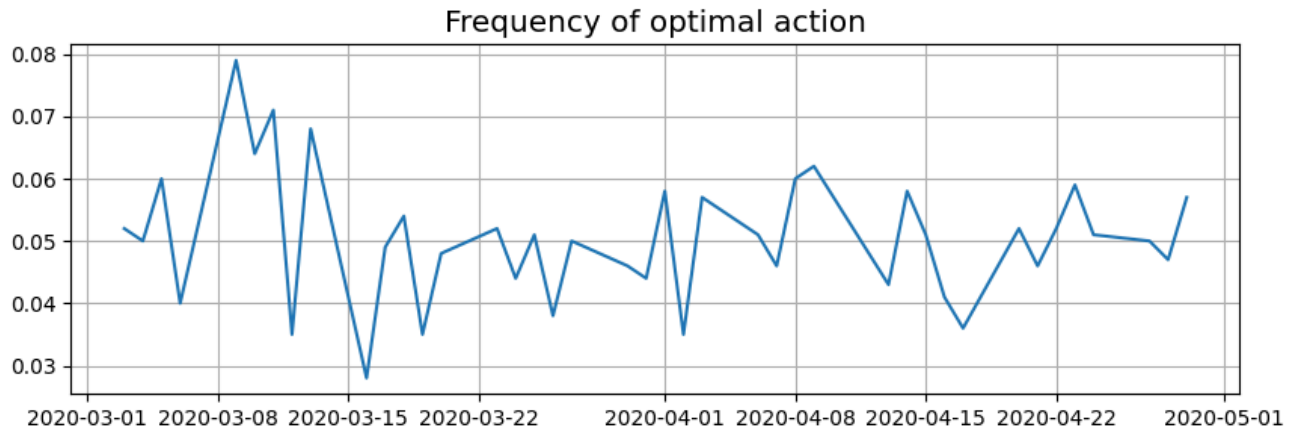
reward_avg = np.zeros((TMAX))
optimal_avg = np.zeros((TMAX))

reward_queue = np.zeros((HOLD, 2))

for run in range(NEPISODES): # noQA E203
    # Initialize q function and actions record
    qvalue = np.zeros((NK))
    qvalue_up = np.zeros((NK))
    nactions = np.zeros((NK))
    for tt in range(TMAX): # noQA E203
        aa_opt = optimal_action(qvalue_up, EPSILON)
        nactions[aa_opt] += 1
        # Compute reward as return over holding period
        reward_queue[HOLD - 1, 0] = (pdata_2[tt + HOLD, aa_opt] - pdata_2[tt, aa_opt]) /
        reward_queue[HOLD - 1, 1] = aa_opt
        # Update Q function using action chosen HOLD days before
        qvalue = reward_update(int(reward_queue[0, 1]), reward_queue[0, 0], qvalue, ALPHA)
        #qvalue = reward_update(int(reward_queue[0, 1]), reward_queue[0, 0], qvalue, 1/na)
        #print(qvalue)
        # Upper-confidence adjustment
        qvalue_up = np.zeros((NK))
        for aa in range(NK):
            # If an action has not been visited, simply give it maximum value across acti
            if nactions[aa] == 0: # noQA E203
                qvalue_up[aa] = np.max(qvalue) + 1.0
            else:
                qvalue_up[aa] = qvalue[aa] + UBC_WEIGHT * np.sqrt(np.log(tt + 1) / nactio
        reward_queue[0 : HOLD - 1, :] = reward_queue[1:HOLD, :] # noQA E203
        reward_avg[tt] += reward_queue[HOLD - 1, 0] / NEPISODES
        optimal_avg[tt] += (aa_opt == np.argmax((pdata_2[tt + HOLD, :] - pdata_2[tt, :]))

plt.plot(pdata_dates_2[HOLD : pdata_2.shape[0]], optimal_avg) # noQA E203
plt.title("Frequency of optimal action", fontsize="x-large")
plt.grid(True)
fig = plt.gcf()
fig.set_size_inches(10, 3)
plt.show()

```

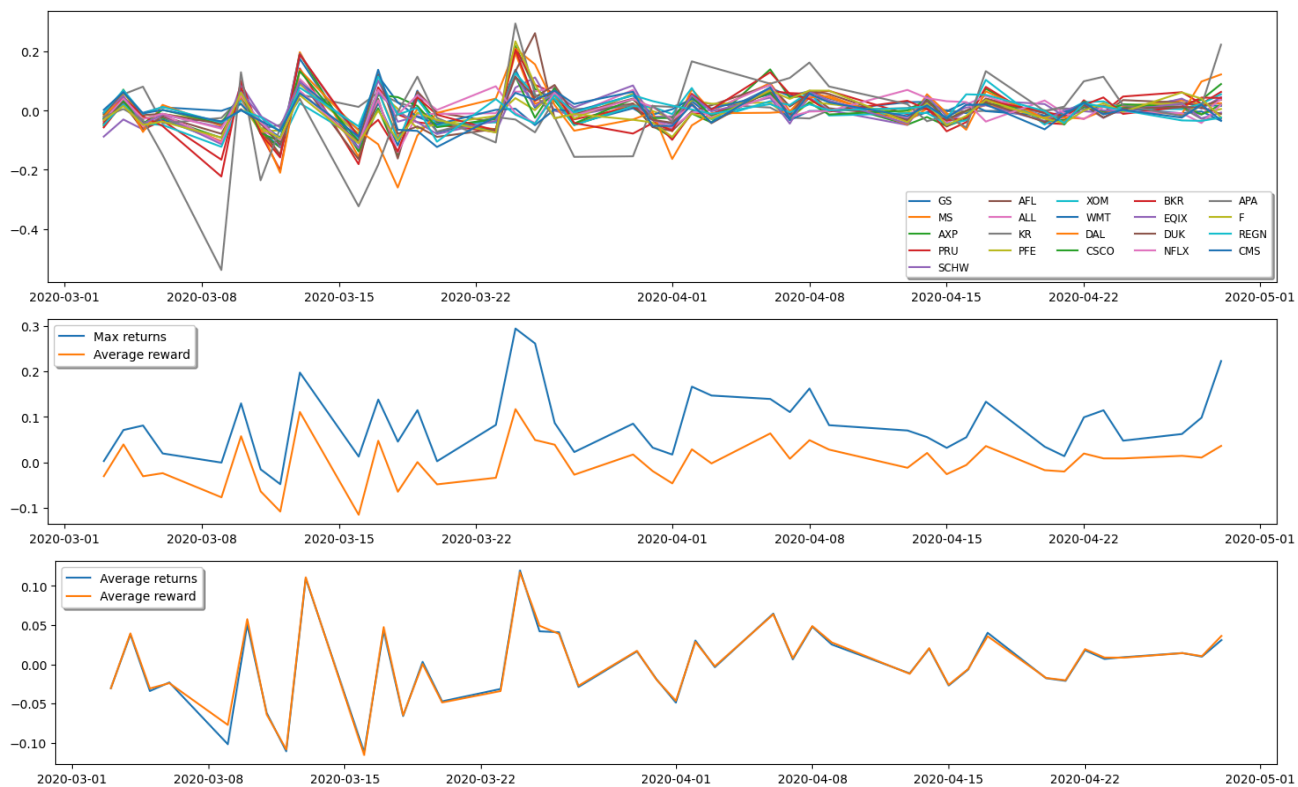


```
plt.plot(pdata_dates_2[HOLD : pdata_2.shape[0]],(pdata_2[HOLD : pdata_2.shape[0], :] - pd
#plt.legend(tickers)
legend = plt.legend(tickers_2, loc="lower right", shadow=True, fontsize="small", ncol=5)
fig = plt.gcf()
fig.set_size_inches(18, 4)
plt.show()
```

```
plt.plot(pdata_dates_2[HOLD : pdata_2.shape[0]],np.max((pdata_2[HOLD : pdata_2.shape[0],
plt.plot(pdata_dates_2[HOLD : pdata_2.shape[0]], reward_avg, label="Average reward") # n
legend = plt.legend(loc="upper left", shadow=True)
fig = plt.gcf()
fig.set_size_inches(18, 3)
plt.show()
```

```
plt.plot(pdata_dates_2[HOLD : pdata_2.shape[0]],np.mean((pdata_2[HOLD : pdata_2.shape[0],
plt.plot(pdata_dates_2[HOLD : pdata_2.shape[0]], reward_avg, label="Average reward") # n
legend = plt.legend(loc="upper left", shadow=True)
fig = plt.gcf()
fig.set_size_inches(18, 3)
plt.show()
```

```
# Average frequency of optimal action
print(np.mean(optimal_avg))
# Average annualized return from holding the equally-weighted portfolio
print((1+ np.mean((pdata_2[HOLD : pdata_2.shape[0], :] - pdata_2[0:TMAX, :]) / pdata_2[0:
# Average annualized return from holding the Bandit portfolio
print((1 + np.mean(reward_avg)) ** (250 / HOLD) - 1,np.sqrt(250 / HOLD) * np.std(reward_a
```



0.05048780487804881

-0.0793966737901296 0.7884441179773355

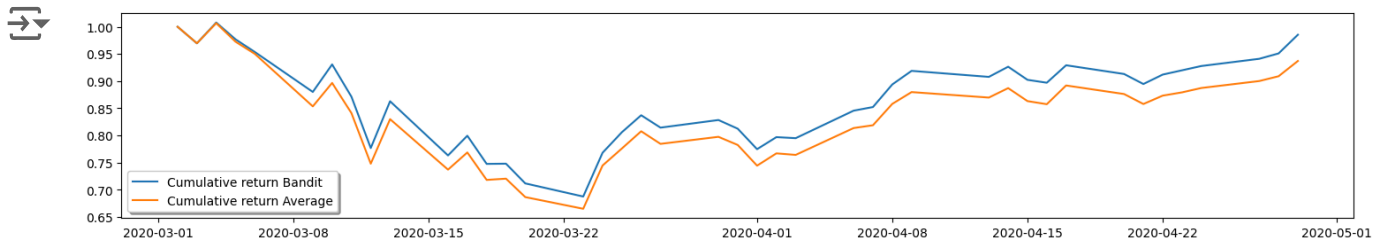
0.23926054112617434 0.7783432350145056

```

return_cumulative = np.zeros((TMAX + 1, 2))
return_cumulative[0, 0] = 1
return_cumulative[0, 1] = 1
for tt in range(1, TMAX + 1): # noQA E203
    return_cumulative[tt, 0] = return_cumulative[tt - 1, 0] * (1 + reward_avg[tt - 1])
    rmean = np.mean((pdata_2[tt + HOLD - 1, :] - pdata_2[tt - 1, :]) / pdata_2[tt - 1, :])
    return_cumulative[tt, 1] = return_cumulative[tt - 1, 1] * (1 + rmean) # noQA E203

plt.plot(pdata_dates_2[HOLD - 1 : pdata_2.shape[0]], return_cumulative[:, 0], label="Cumula
plt.plot(pdata_dates_2[HOLD - 1 : pdata_2.shape[0]], return_cumulative[:, 1], label="Cumula
legend = plt.legend(loc="lower left", shadow=True)
fig = plt.gcf()
fig.set_size_inches(18, 3)
plt.show()

```



✓ 1. Epsilon-Greedy Algorithm on the Updated 30 Securities

```

# Bandit problem for stock selection
NK = pdata.shape[1]
EPSILON = 0.15
ALPHA = 0.5
NEPISODES = 1000
HOLD = 1
TMAX = pdata.shape[0] - HOLD

seed(1234)

reward_avg = np.zeros((TMAX))
optimal_avg = np.zeros((TMAX))

reward_queue = np.zeros((HOLD, 2))

for run in range(NEPISODES):
    # Initialize q function and actions record
    qvalue = np.zeros((NK))
    nactions = np.zeros((NK))
    for tt in range(TMAX):
        aa_opt = optimal_action(qvalue, EPSILON)
        nactions[aa_opt] += 1
        # Compute reward as return over holding period
        reward_queue[HOLD - 1, 0] = (pdata[tt + HOLD, aa_opt] - pdata[tt, aa_opt]) / pdat
        reward_queue[HOLD - 1, 1] = aa_opt
        # Update Q function using action chosen HOLD days before
        qvalue = reward_update(int(reward_queue[0, 1]), reward_queue[0, 0], qvalue, ALPHA)
        reward_queue[0 : HOLD - 1, :] = reward_queue[1:HOLD, :]
        reward_avg[tt] += reward_queue[HOLD - 1, 0] / NEPISODES
        optimal_avg[tt] += (aa_opt == np.argmax((pdata[tt + HOLD, :] - pdata[tt, :])) / pd

```

Start coding or [generate](#) with AI.

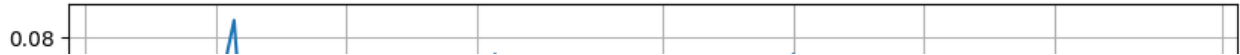
```

plt.plot(pdata_dates[HOLD : pdata.shape[0]], optimal_avg) # noQA E203
plt.title("Frequency of optimal action", fontsize="x-large")
plt.grid(True)
fig = plt.gcf()
fig.set_size_inches(10, 3)
plt.show()

```



Frequency of optimal action



```
plt.plot(pdata_dates[HOLD : pdata.shape[0]],(pdata[HOLD : pdata.shape[0], :] - pdata[0:TMA]
#plt.legend(tickers)
legend = plt.legend(tickers, loc="lower right", shadow=True, fontsize="small", ncol=5)
fig = plt.gcf()
fig.set_size_inches(18, 4)
plt.show()
```

```
plt.plot(pdata_dates[HOLD : pdata.shape[0]],np.max((pdata[HOLD : pdata.shape[0], :] - pdata
plt.plot(pdata_dates[HOLD : pdata.shape[0]], reward_avg, label="Average reward") # noQA E
legend = plt.legend(loc="upper left", shadow=True)
fig = plt.gcf()
fig.set_size_inches(18, 3)
plt.show()
```

```
plt.plot(pdata_dates[HOLD : pdata.shape[0]],np.mean((pdata[HOLD : pdata.shape[0], :] - pda
plt.plot(pdata_dates[HOLD : pdata.shape[0]], reward_avg, label="Average reward") # noQA E
legend = plt.legend(loc="upper left", shadow=True)
fig = plt.gcf()
fig.set_size_inches(18, 3)
plt.show()
```

```
# Average frequency of optimal action
print(np.mean(optimal_avg))
# Average annualized return from holding the equally-weighted portfolio
print((1+ np.mean((pdata[HOLD : pdata.shape[0], :] - pdata[0:TMAX, :]) / pdata[0:TMAX, :]))
# Average annualized return from holding the Bandit portfolio
print((1 + np.mean(reward_avg)) ** (250 / HOLD) - 1,np.sqrt(250 / HOLD) * np.std(reward_avg)
```

