TSM

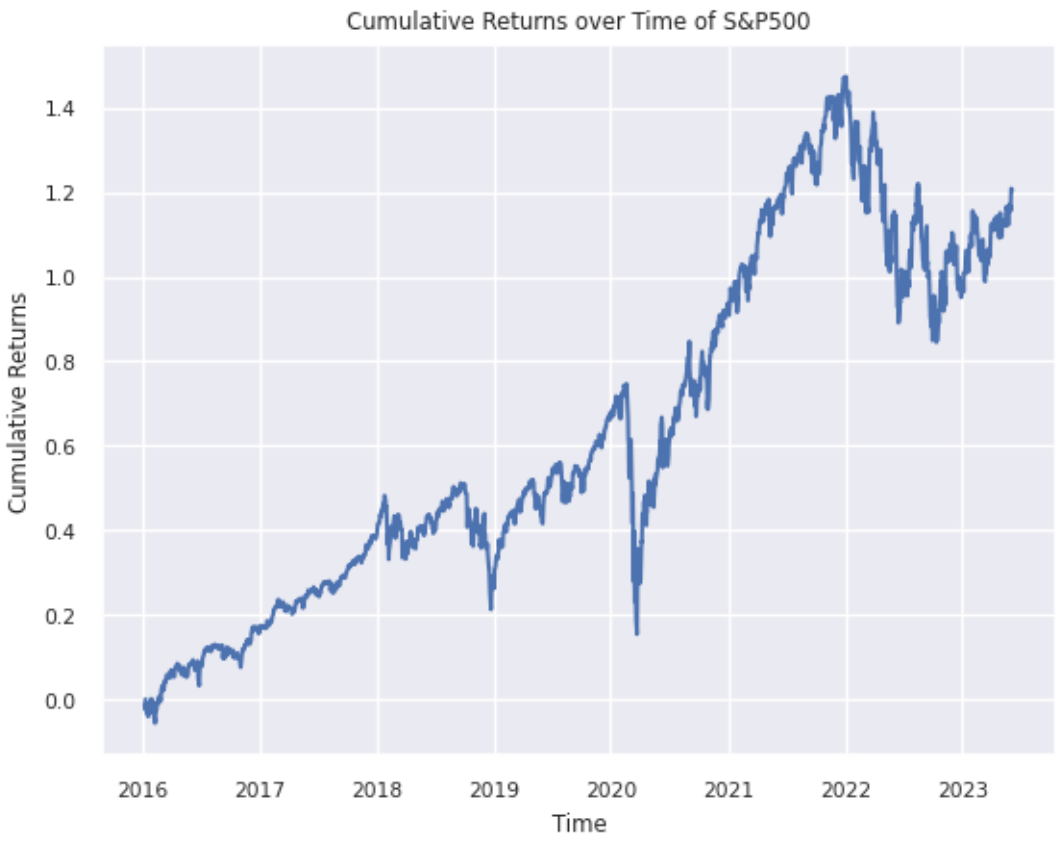
June 14, 2023

[ ]:
```
import pandas as pd
import numpy as np
import requests
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
from matplotlib.lines import Line2D
from io import BytesIO
from pandas.tseries.offsets import MonthEnd
import pandas_datareader



# URL of your Excel file
url = 'https://docs.google.com/spreadsheets/d/e/
 ↪2PACX-1vRIXmPmhHHetEyxK-PMz1uyTzBcN2s9-6noA-SdGidWlnqTj1OgzQm7Ekn68D6ASg/pub?
 ↪output=xlsx'

# Send a GET request to the URL
response = requests.get(url)

# Read the content of the response with pandas
data = pd.read_excel(BytesIO(response.content), sheet_name=None)

# Now 'data' is a dictionary where the keys are the names of the sheets and the␣
 ↪values are the dataframes
# You can access the dataframe of a specific sheet like this:
SnP500 = data['S&P500']
dowjones = data['Dow Jones Industrial Average']
TSX = data['TSX']
UKFTSE = data['UK FTSE']
ITALYftse = data['ITALY FTSE MIB']
asx = data['Australia ASX']
cac = data['CAC']
ibex = data['SPAIN IBEX']
Gold = data['Gold']
Coffee = data['Coffee']
```

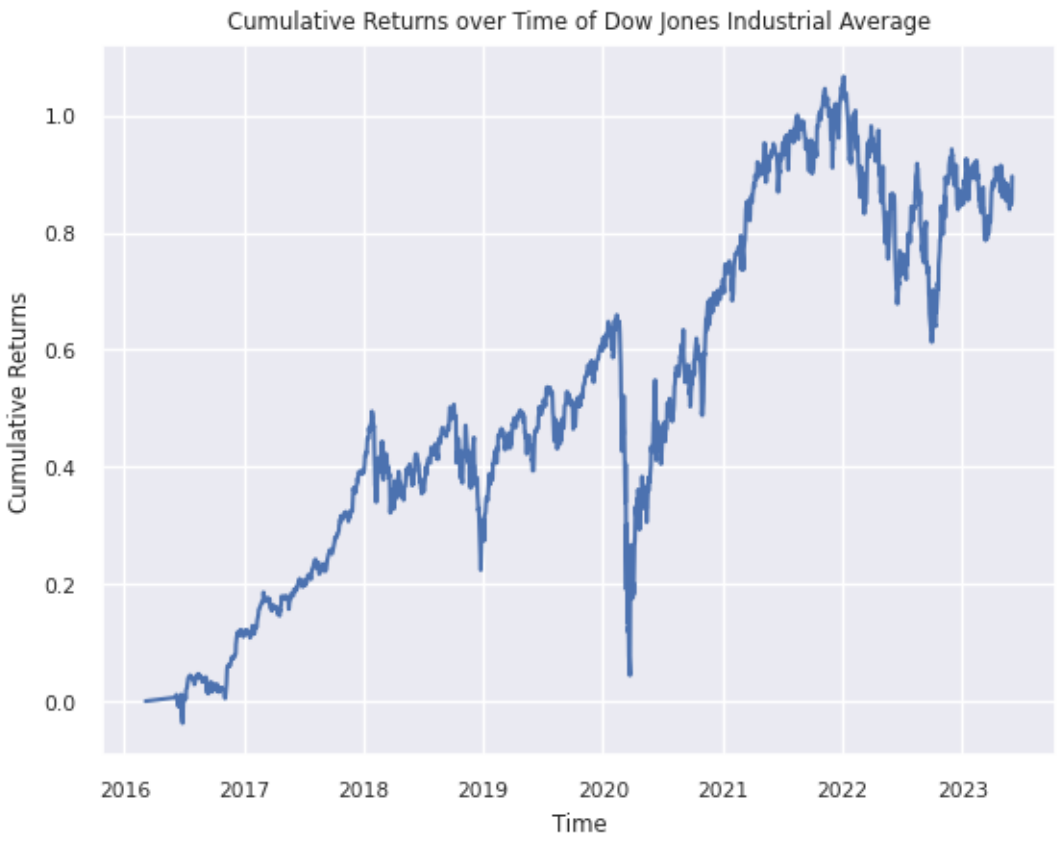
```
Wheat = data['Wheat']
Aluminum = data['Aluminum']
BRENTOIL = data['BRENTOIL']
ICESUGAR = data['ICESUGAR']
EURUSD = data['EURUSD']
GBPUSD = data['GBPUSD']
USDCAD = data['USDCAD']
AUDUSD = data['AUDUSD']
USDJPY = data['USDJPY']
```

[ ]:
```
def analyze_dataframe(df1, df_name, plot):
    df=df1.copy()
    df.dropna()
    # Ensure 'Date' column is in datetime format
    df['Date'] = pd.to_datetime(df['Date'])
    df.set_index('Date', inplace=True)
    df = df.sort_values(by='Date')
    # Calculate daily returns
    df['Daily Returns'] = df['Price'].pct_change()

    # Define momentum signal function and apply it
    def momentum_signal(data, lookback_period):
        return np.sign(data.shift(lookback_period).mean())

    lookback_periods = [21, 63, 252]  # Approximate 1-month, 3-months, and␣
↪12-months in trading days
    for period in lookback_periods:
        df[f'{period}D Momentum Signal'] = momentum_signal(df['Daily Returns'],␣
↪period)

    # Average the signals
    df['Momentum Signal'] = df[[f'{period}D Momentum Signal' for period in␣
↪lookback_periods]].mean(axis=1)

    # Compute strategy returns without volatility scaling
    df['Momentum Returns'] = df['Momentum Signal'].shift() * df['Daily Returns']

    # Transaction cost and fees
    df['Transaction Costs'] = 0.0000  # This is a placeholder
    df['Fees'] = 0.0000 + 000.000 * np.maximum(df['Momentum Returns'] -␣
↪df['Transaction Costs'], 0)
    df['Net Momentum Returns'] = df['Momentum Returns'] - df['Transaction␣
↪Costs'] - df['Fees']

    # Calculate annualized returns
    df['Annualized Returns'] = df['Net Momentum Returns'].mean() * 252
```

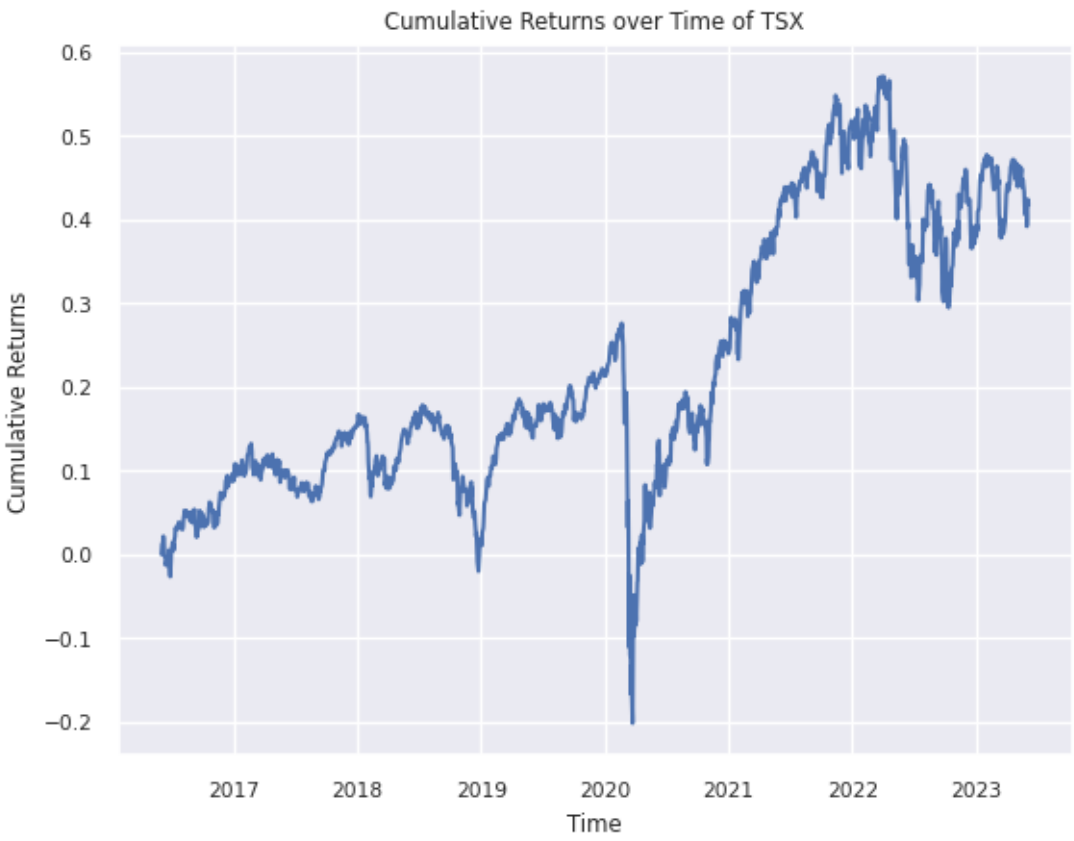
```
    # Calculate annualized volatility
    df['Annualized Volatility'] = df['Net Momentum Returns'].std() * np.
 ↪sqrt(252)

    # Calculate Sharpe ratio
    df['Sharpe Ratio'] = df['Annualized Returns'] / df['Annualized Volatility']

    # Calculate t-statistic
    df['t-statistic'] = stats.ttest_1samp(df['Net Momentum Returns'].dropna(),␣
 ↪0)[0]

    # Create a DataFrame to hold the results
    results = pd.DataFrame({
        'Asset': [df_name],
        'Annualized Excess Return': [df['Annualized Returns'].iloc[-1]],
        'Volatility': [df['Annualized Volatility'].iloc[-1]],
        'Sharpe Ratio': [df['Sharpe Ratio'].iloc[-1]],
        't-statistic': [df['t-statistic'].iloc[-1]]
    })

    if plot == True:
      # Plot the cumulative returns
      cumulative_returns = (1 + df['Net Momentum Returns'].fillna(0)).cumprod()␣
 ↪- 1
      plt.plot(cumulative_returns)
      plt.xlabel('Time')
      plt.ylabel('Cumulative Returns')
      plt.title(f'Cumulative Returns over Time of {df_name}')
      plt.show()


    return results

# Create an empty DataFrame to hold all the results
all_results = pd.DataFrame()

products = ['S&P500', 'Dow Jones Industrial Average', 'TSX', 'UK FTSE',
            'ITALY FTSE MIB', 'Australia ASX', 'CAC', 'SPAIN IBEX', 'Gold',␣
 ↪'Coffee', 'Wheat', 'Aluminum', 'BRENTOIL', 'BRENTOIL', 'ICESUGAR', 'EURUSD',␣
 ↪'GBPUSD', 'USDCAD', 'AUDUSD', 'USDJPY']
for i in products:
  result = analyze_dataframe(data[i], i, True)
  all_results = all_results.append(result, ignore_index=True)

# Now, all_results DataFrame contains the annualized returns, volatility,␣
 ↪Sharpe ratio, and t-statistic for all assets.
print(all_results)
```



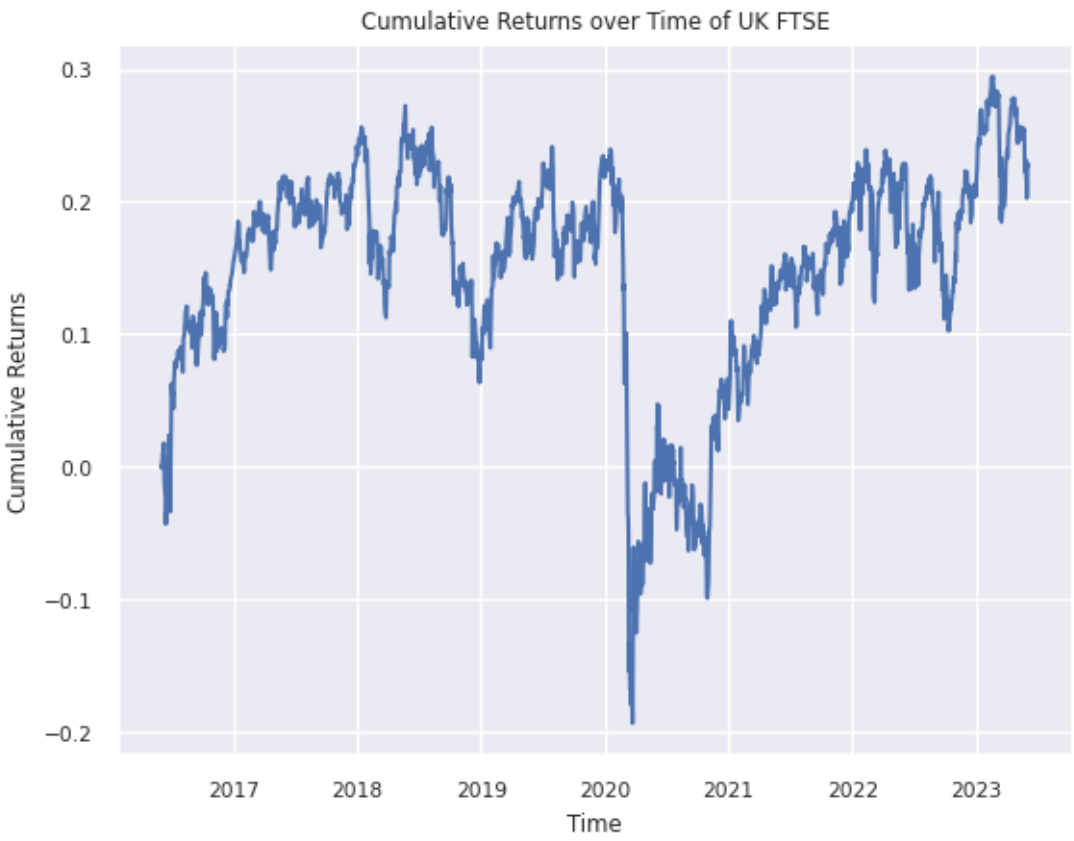

```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```



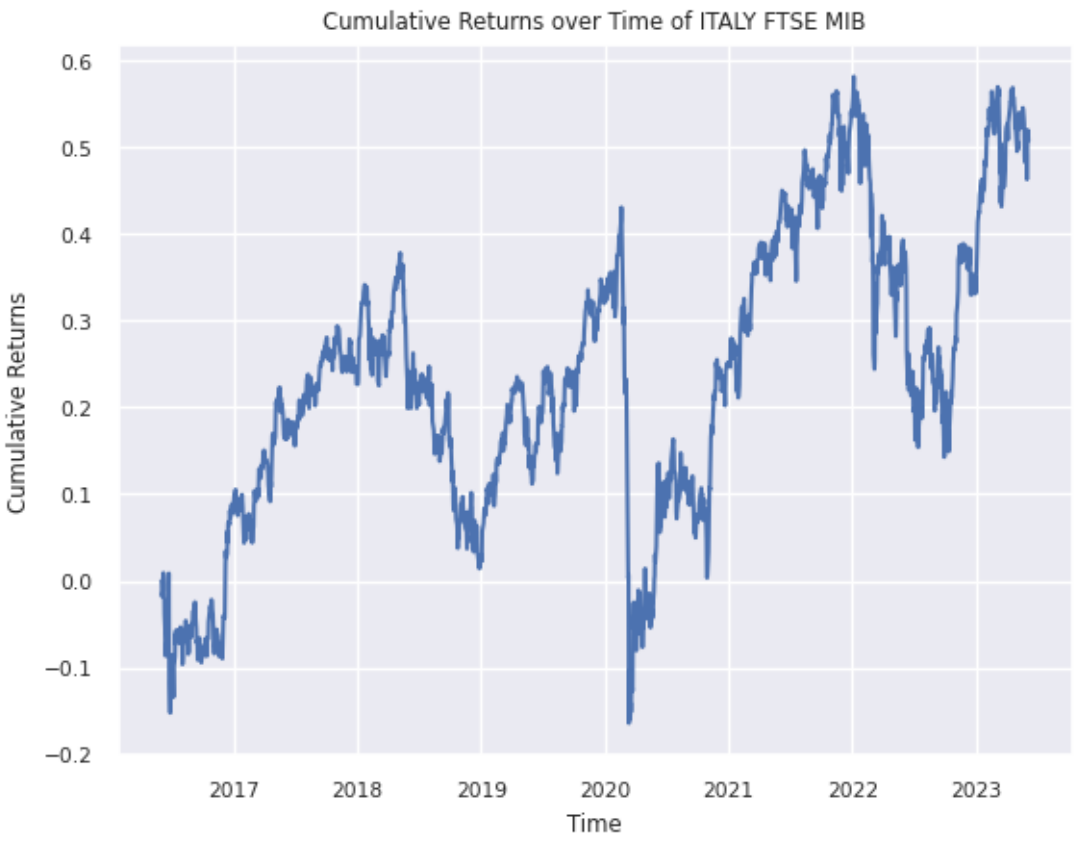

```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```




```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```

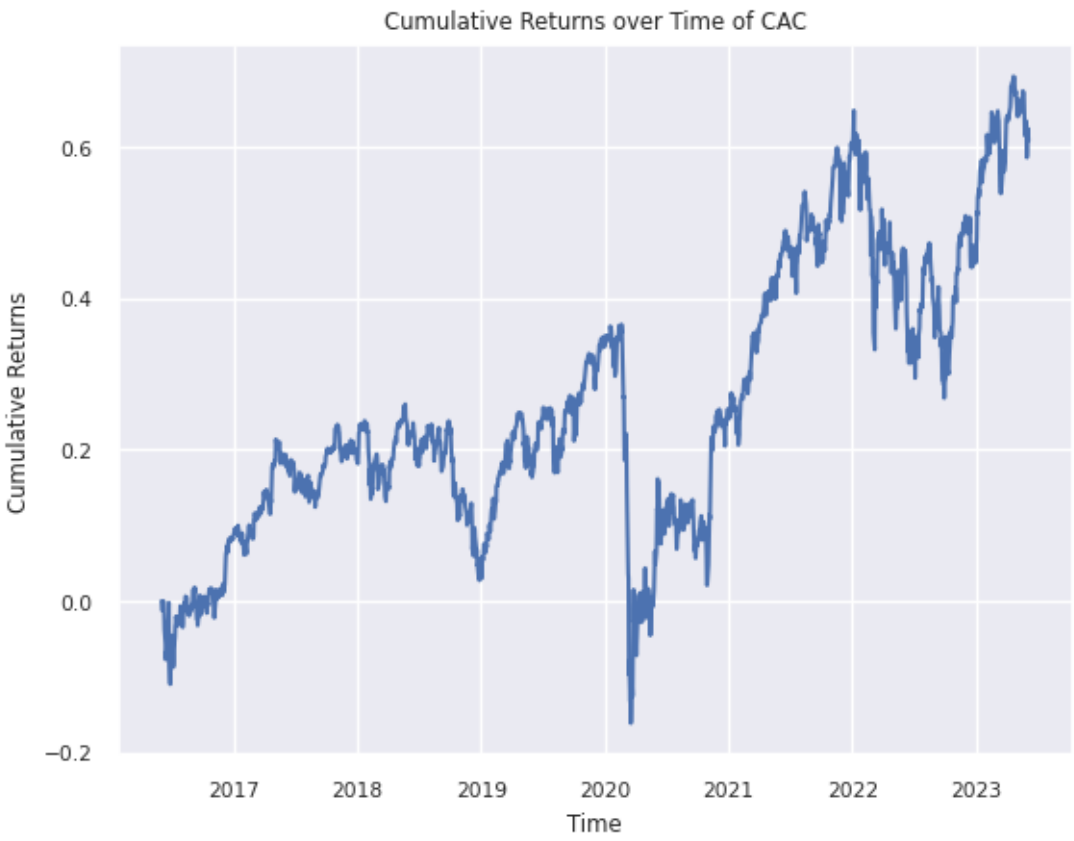



```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```

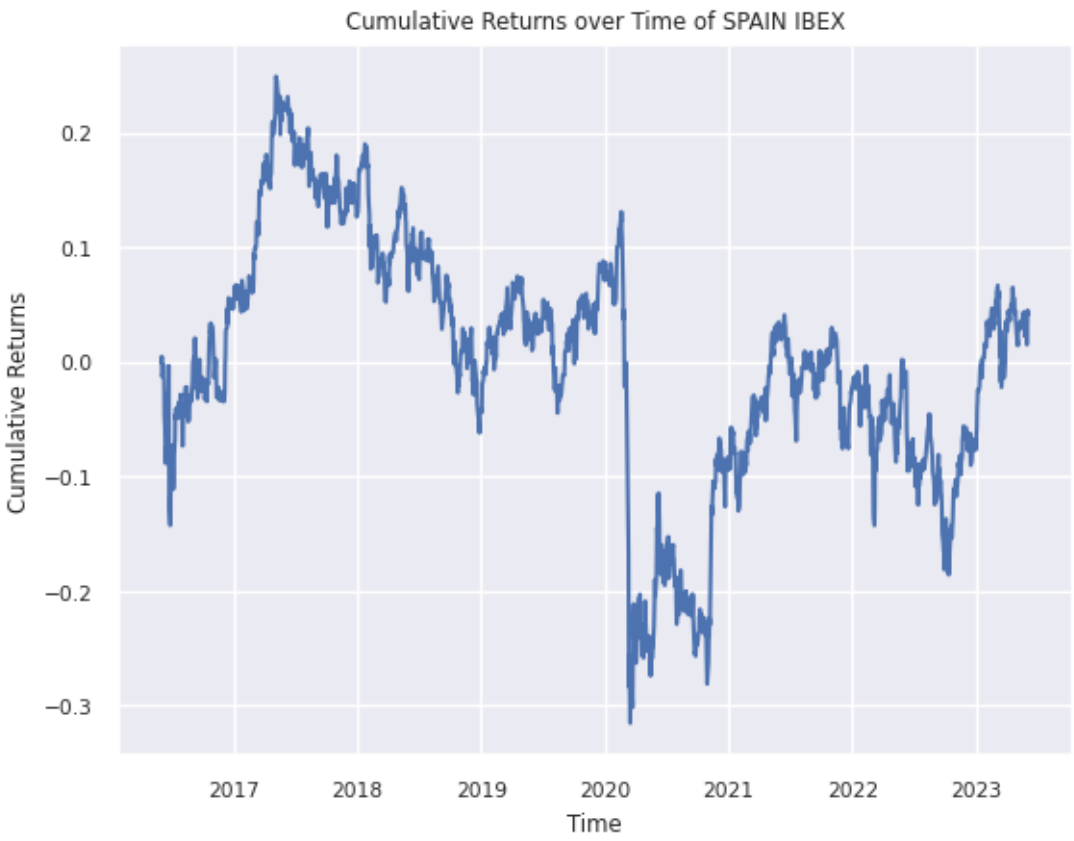



```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```

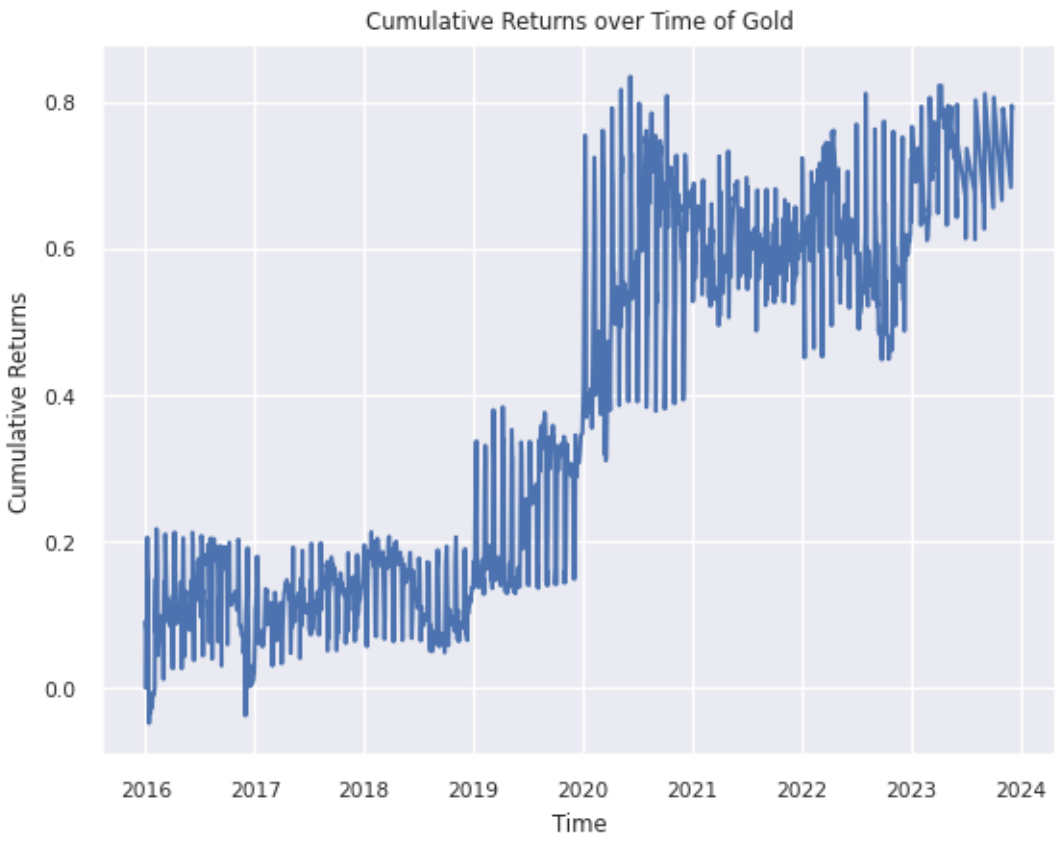

```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```



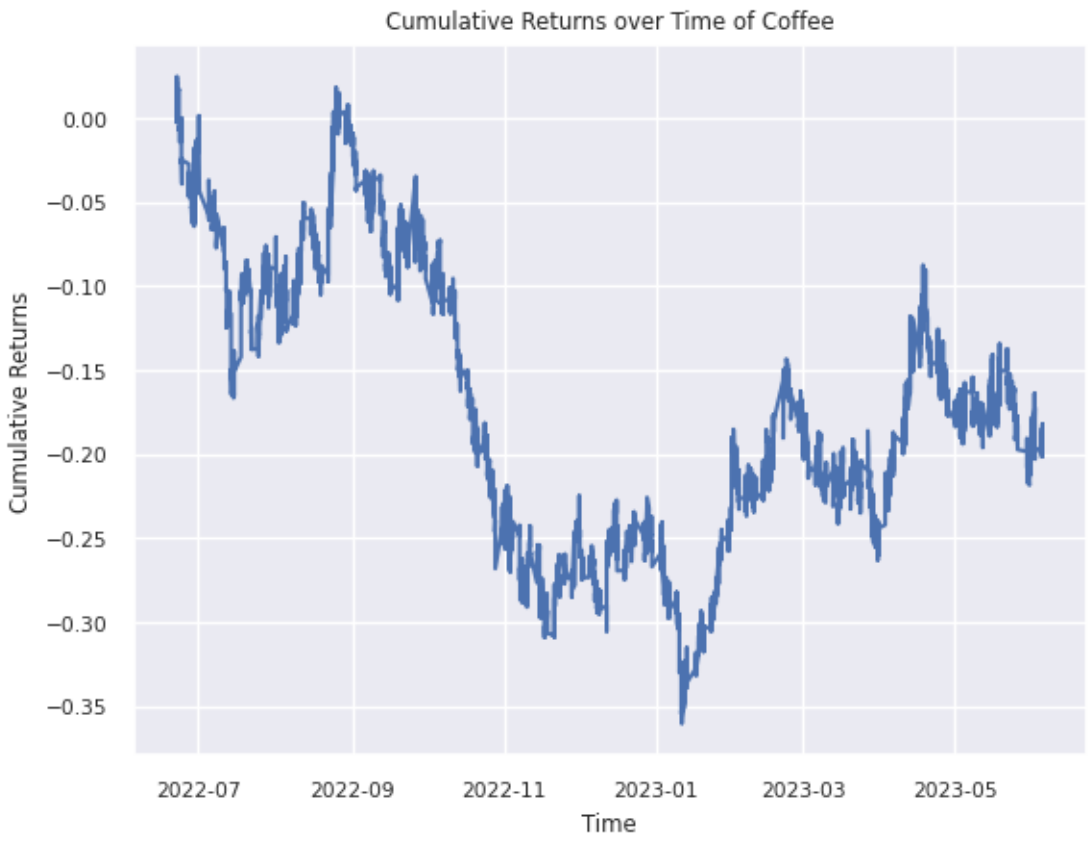

```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```



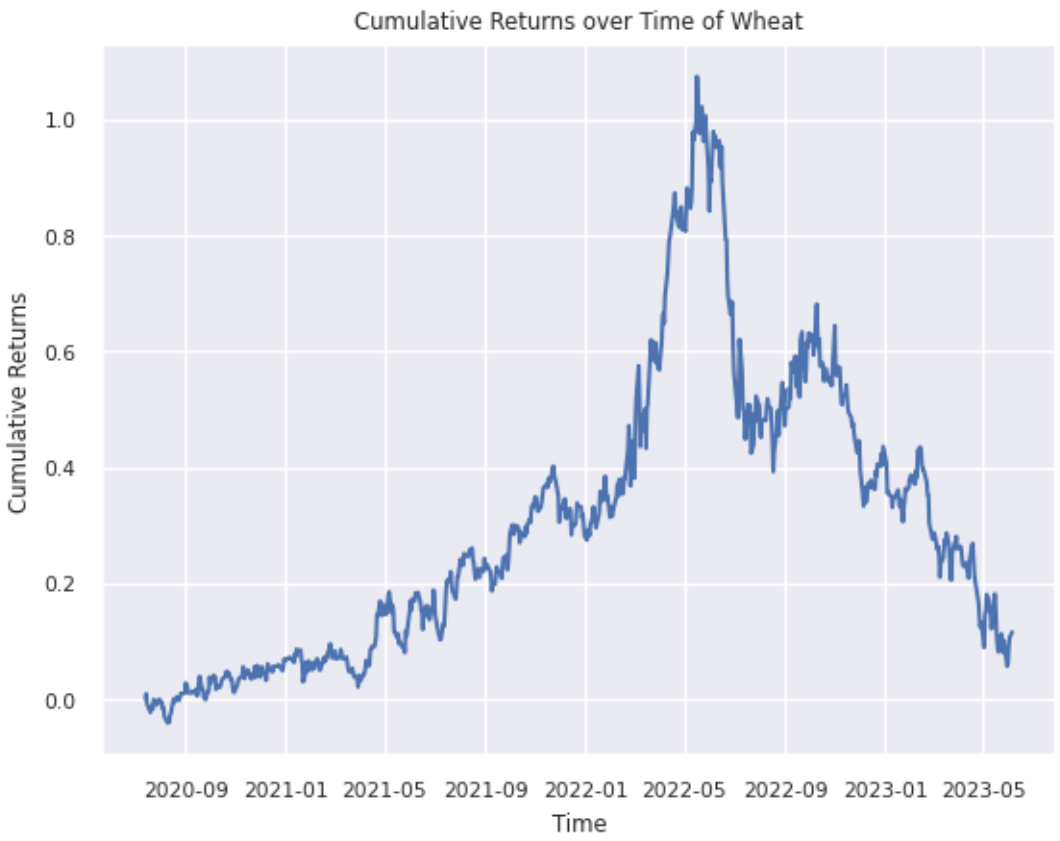

```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```



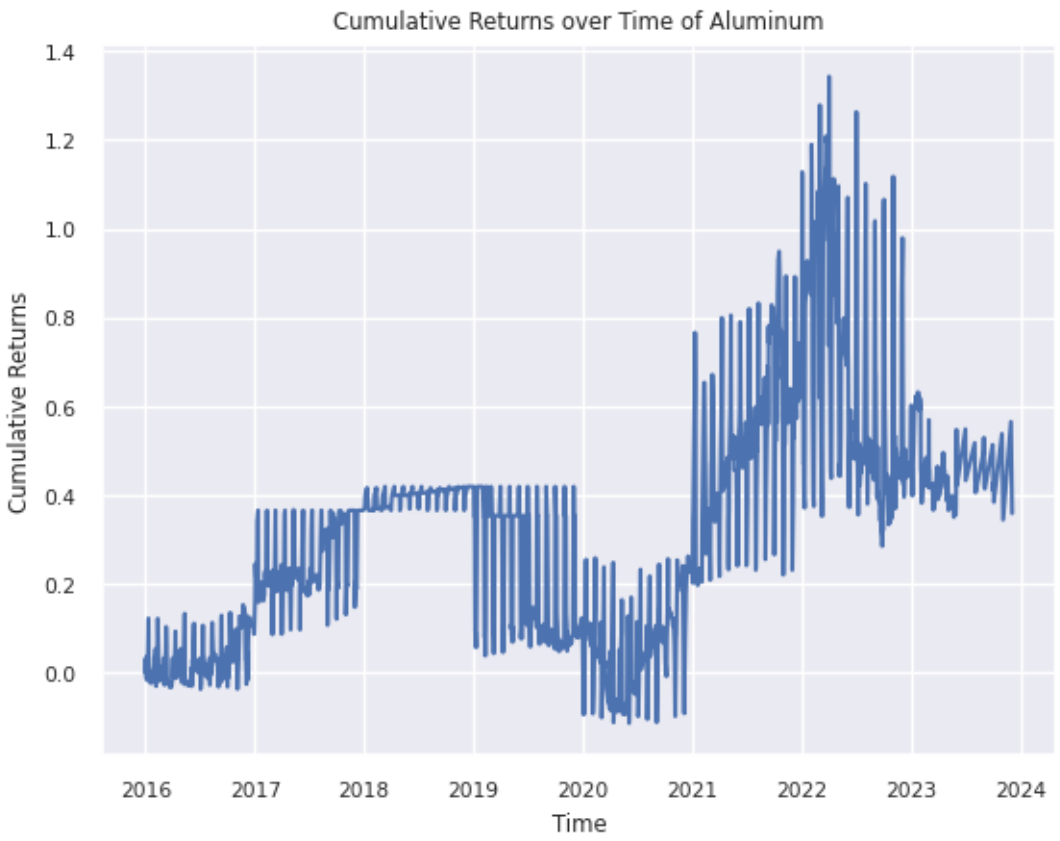

```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```



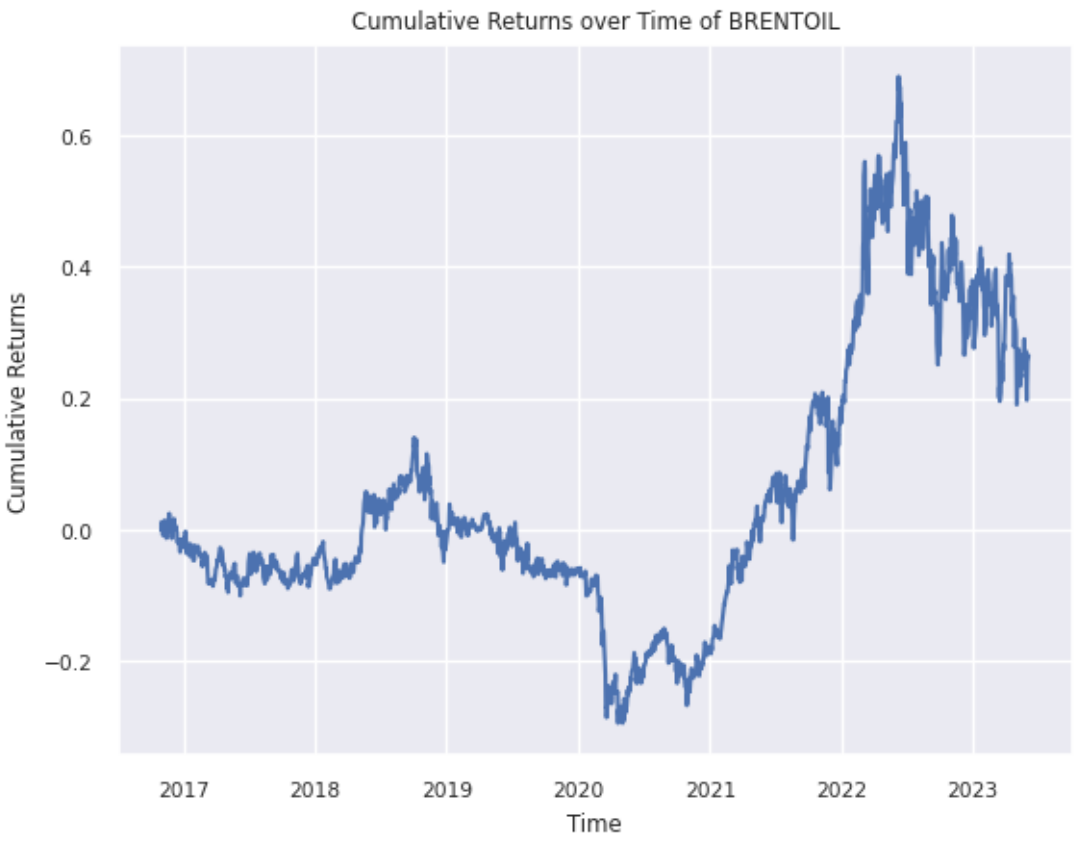

```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```



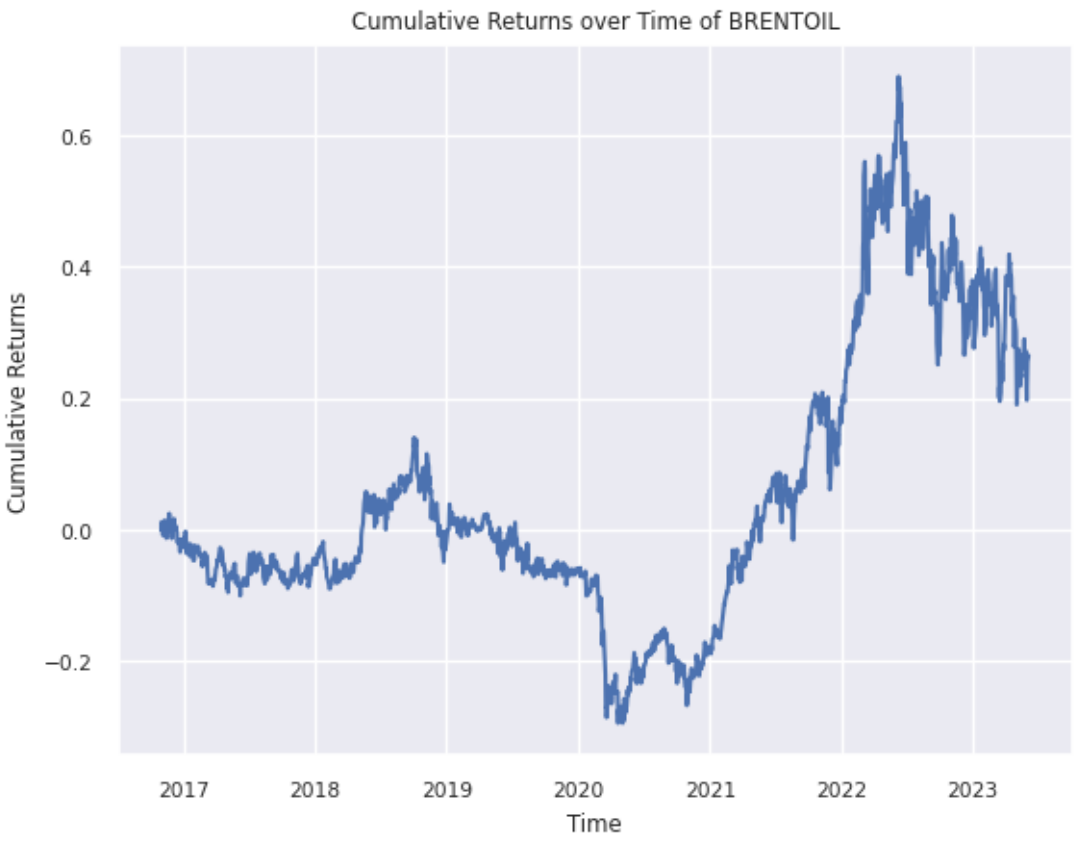

```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```



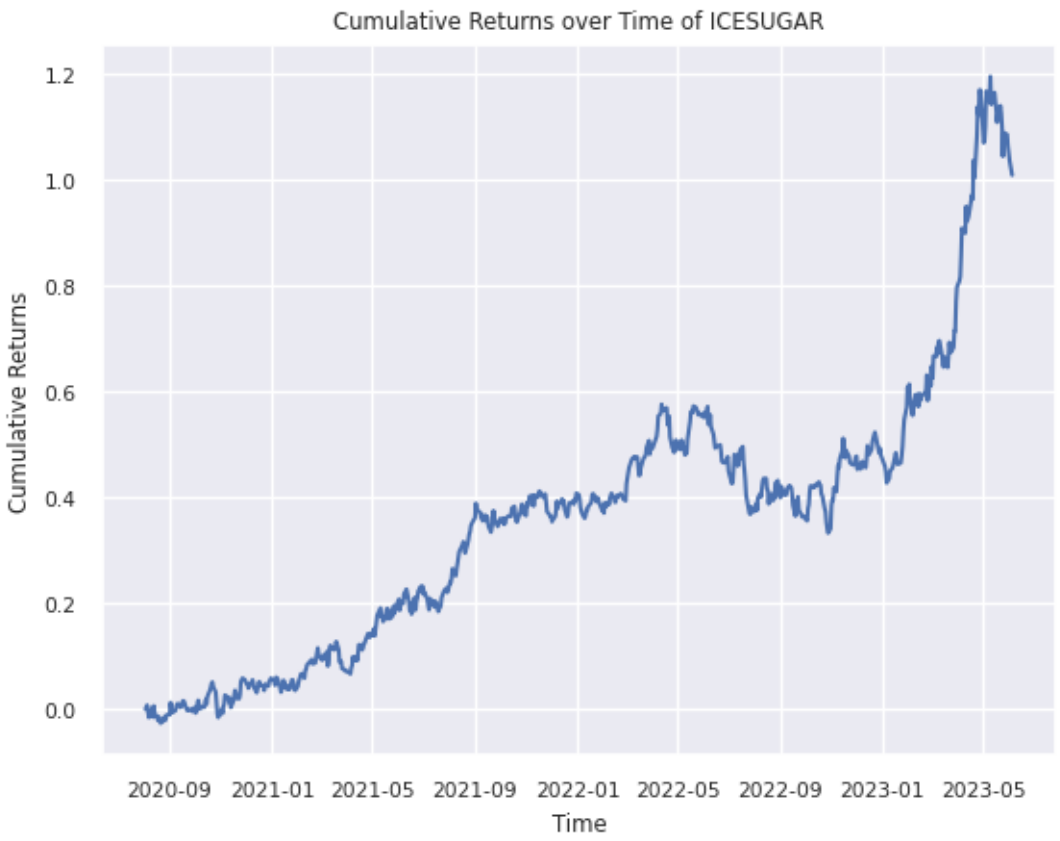

```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```



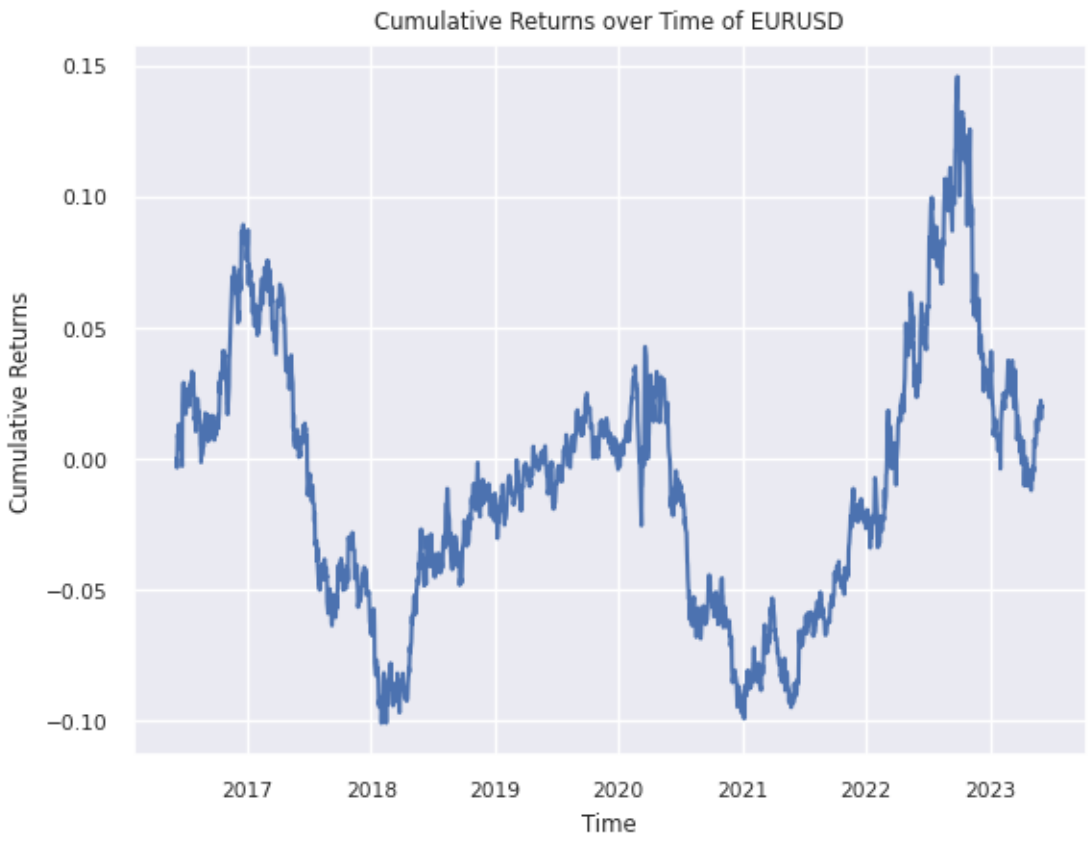

```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```



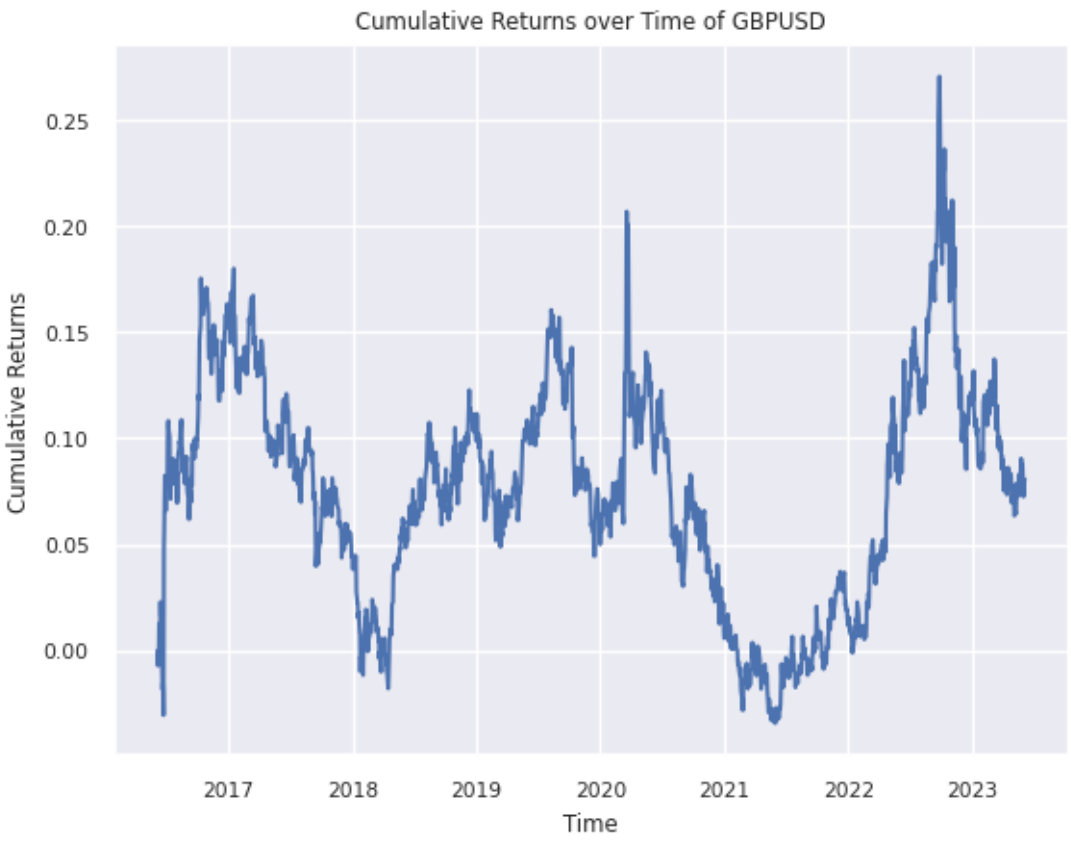

```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```



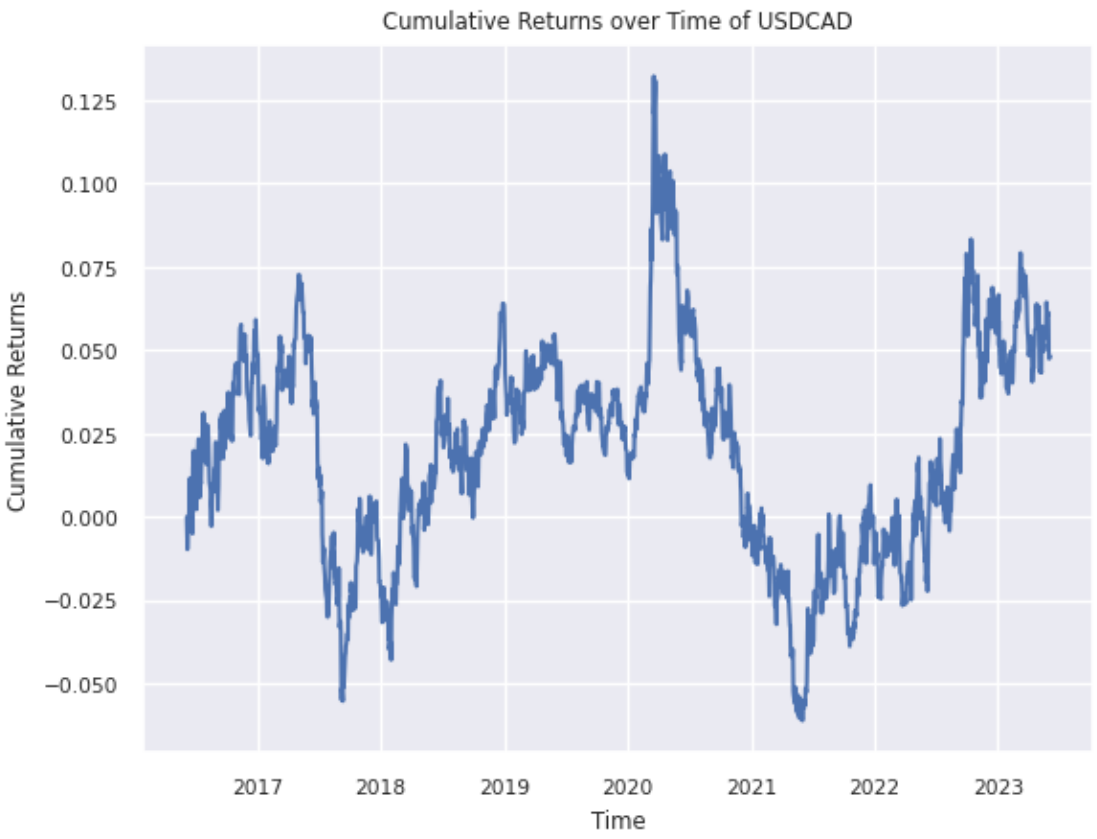

```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```



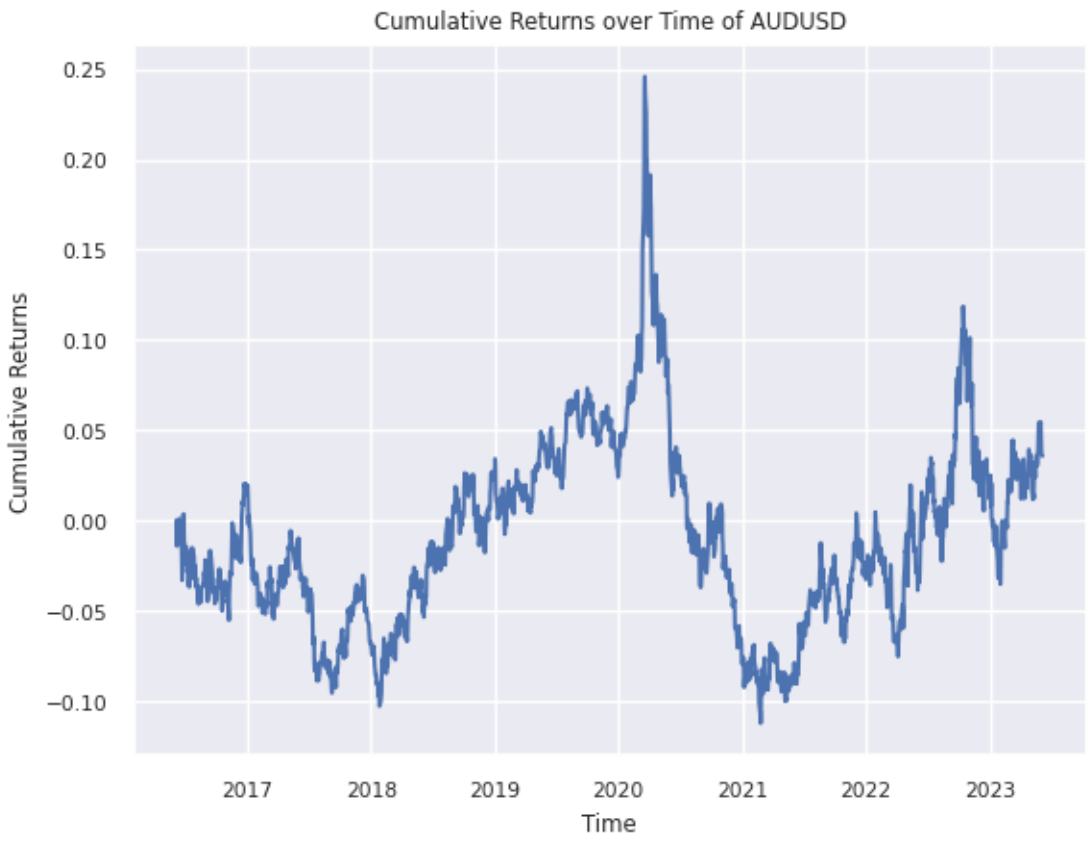

```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```



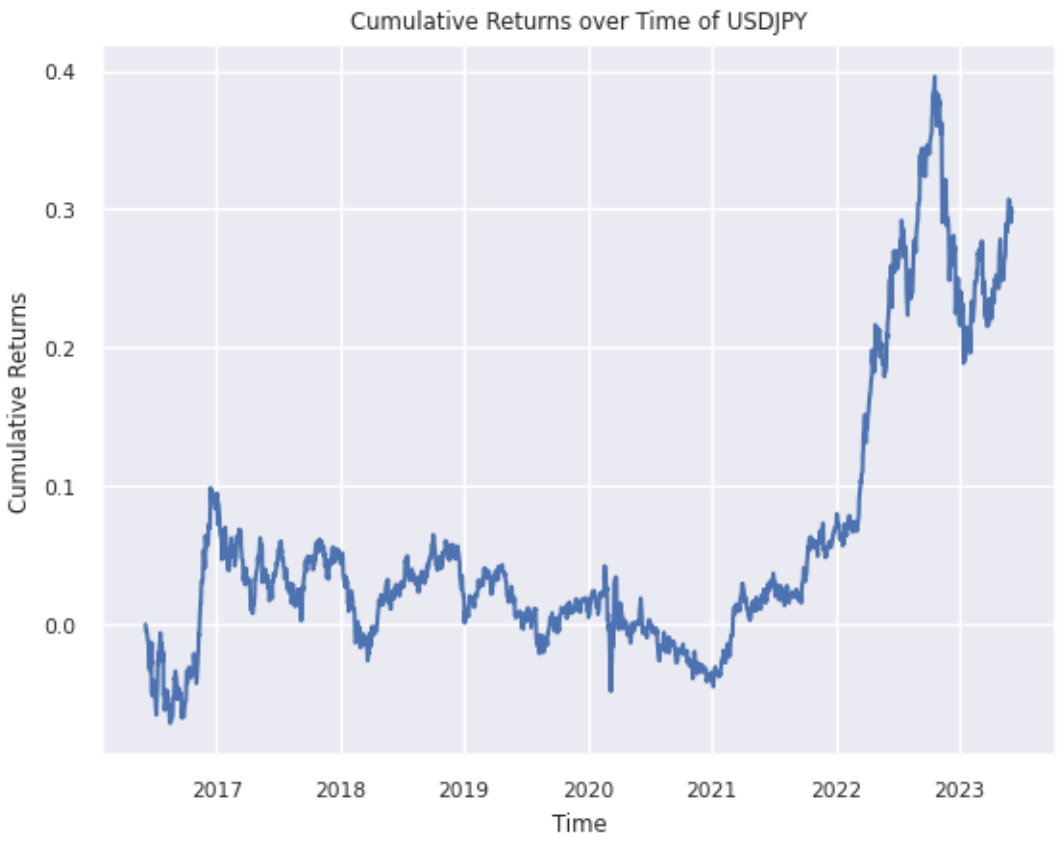

```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```




```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```




```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```



	Asset	Annualized Excess Return	Volatility	\
0	S&P500	0.125364	0.190717	
1	Dow Jones Industrial Average	0.109935	0.191133	
2	TSX	0.062872	0.159732	
3	UK FTSE	0.042516	0.162781	
4	ITALY FTSE MIB	0.083047	0.221664	
5	Australia ASX	0.055803	0.157896	
6	CAC	0.084916	0.190000	
7	SPAIN IBEX	0.025595	0.198037	
8	Gold	0.253914	0.592132	
9	Coffee	0.002634	0.244012	
10	Wheat	0.072735	0.264742	
11	Aluminum	0.521761	0.987115	
12	BRENTOIL	0.059915	0.223222	
13	BRENTOIL	0.059915	0.223222	
14	ICESUGAR	0.259055	0.160553	
15	EURUSD	0.005352	0.073111	
16	GBPUSD	0.015675	0.099939	
17	USDCAD	0.009087	0.071918	
18	AUDUSD	0.009785	0.099562	

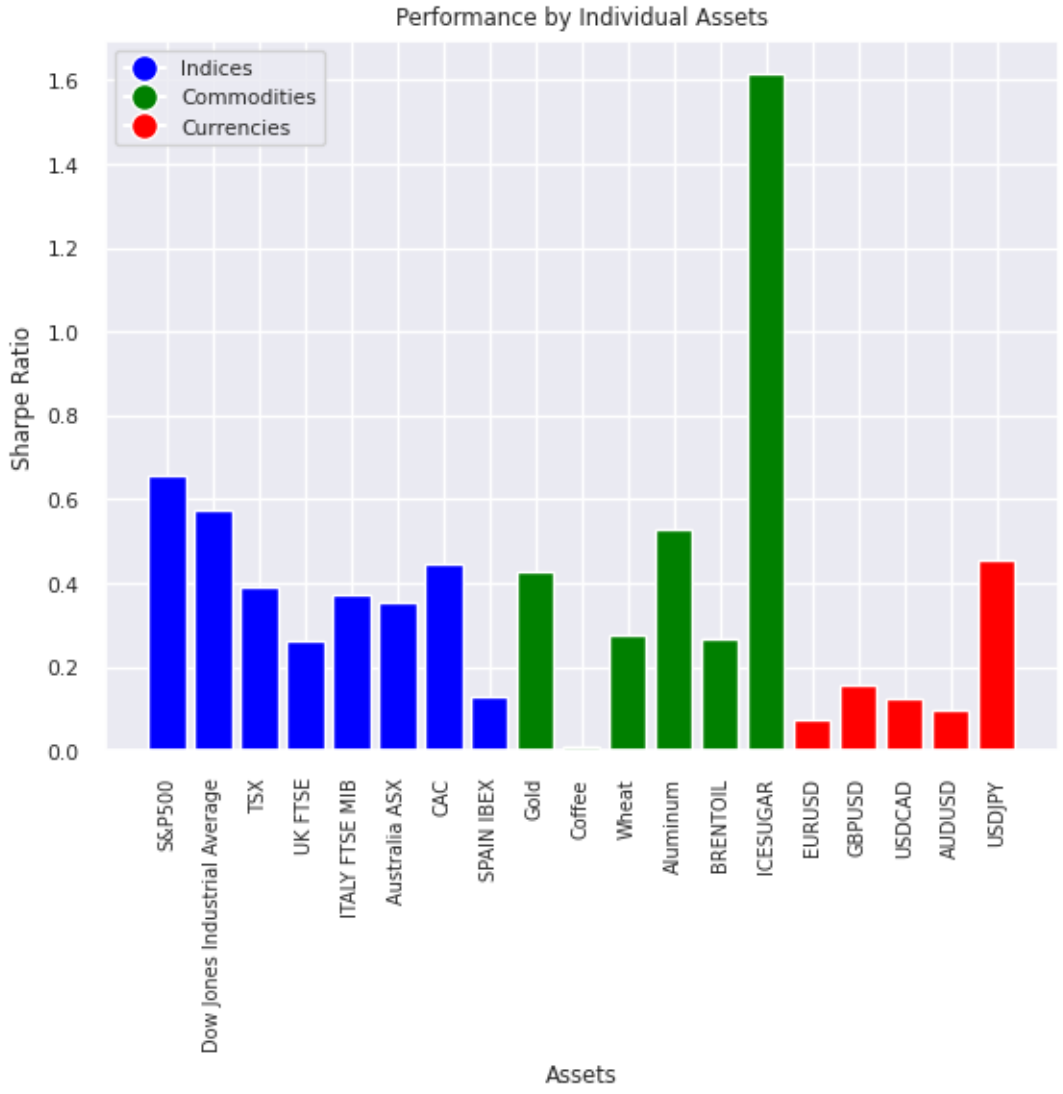
```
19                          USDJPY                    0.039764     0.087009
```

```
    Sharpe Ratio  t-statistic
0       0.657330     1.786309
1       0.575175     1.520476
2       0.393610     1.039621
3       0.261185     0.692010
4       0.374651     0.996558
5       0.353415     0.937431
6       0.446927     1.193135
7       0.129242     0.344838
8       0.428814     1.166249
9       0.010796     0.029793
10      0.274738     0.467606
11      0.528571     1.436790
12      0.268411     0.697762
13      0.268411     0.697762
14      1.613513     2.715947
15      0.073201     0.196992
16      0.156842     0.422078
17      0.126348     0.340016
18      0.098281     0.264484
19      0.457015     1.229878
```

```
<ipython-input-270-636dbf299cb4>:70: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  all_results = all_results.append(result, ignore_index=True)
```

[ ]:
```
# Given data
colors = ['blue', 'blue', 'blue', 'blue', 'blue', 'blue', 'blue', 'blue',␣
 ↪'green', 'green', 'green', 'green', 'green', 'green', 'green', 'red', 'red',␣
 ↪'red', 'red', 'red']
# Create a bar plot
bars = plt.bar(all_results['Asset'], all_results['Sharpe Ratio'], color=colors)

# Create a color legend
legend_elements = [Line2D([0], [0], marker='o', color='w', label='Indices',
                          markerfacecolor='blue', markersize=10),
                   Line2D([0], [0], marker='o', color='w', label='Commodities',
                          markerfacecolor='green', markersize=10),
                   Line2D([0], [0], marker='o', color='w', label='Currencies',
                          markerfacecolor='red', markersize=10)]

plt.legend(handles=legend_elements, loc='upper left')
plt.title('Performance by Individual Assets')
plt.xlabel('Assets')
plt.ylabel('Sharpe Ratio')
```

```
plt.xticks(rotation=90)
plt.xticks(fontsize=7)
plt.figure(figsize=(10,6))

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




```
<Figure size 1000x600 with 0 Axes>
```

[ ]:
```
# Clean the data outside of the main function
Gold['Date'] = pd.to_datetime(Gold['Date'])
Aluminum['Date'] = pd.to_datetime(Aluminum['Date'])
```

```
# Create a dataframe with all the assets
returns_df = SnP500.rename(columns={'Price': 'S&P500'}).merge(
    dowjones.rename(columns={'Price': 'dowjones'}), on='Date').merge(
    TSX.rename(columns={'Price': 'TSX'}), on='Date').merge(
    UKFTSE.rename(columns={'Price': 'UKFTSE'}), on='Date').merge(
    ITALYftse.rename(columns={'Price': 'ITALYftse'}), on='Date').merge(
    asx.rename(columns={'Price': 'asx'}), on='Date').merge(
    cac.rename(columns={'Price': 'cac'}), on='Date').merge(
    ibex.rename(columns={'Price': 'ibex'}), on='Date').merge(
    Gold.rename(columns={'Price': 'Gold'}), on='Date').merge(
    Wheat.rename(columns={'Price': 'Wheat'}), on='Date').merge(
    Aluminum.rename(columns={'Price': 'Aluminum'}), on='Date').merge(
    BRENTOIL.rename(columns={'Price': 'BRENTOIL'}), on='Date').merge(
    ICESUGAR.rename(columns={'Price': 'ICESUGAR'}), on='Date').merge(
    EURUSD.rename(columns={'Price': 'EURUSD'}), on='Date').merge(
    GBPUSD.rename(columns={'Price': 'GBPUSD'}), on='Date').merge(
    USDCAD.rename(columns={'Price': 'USDCAD'}), on='Date').merge(
    AUDUSD.rename(columns={'Price': 'AUDUSD'}), on='Date').merge(
    USDJPY.rename(columns={'Price': 'USDJPY'}), on='Date')
returns_df.set_index('Date', inplace=True)

# Compute the returns for each asset
returns = returns_df.pct_change().dropna()

# Create the correlation matrix for all the assets
corr_matrix = returns.corr().round(3)
corr_matrix
```

[ ]:

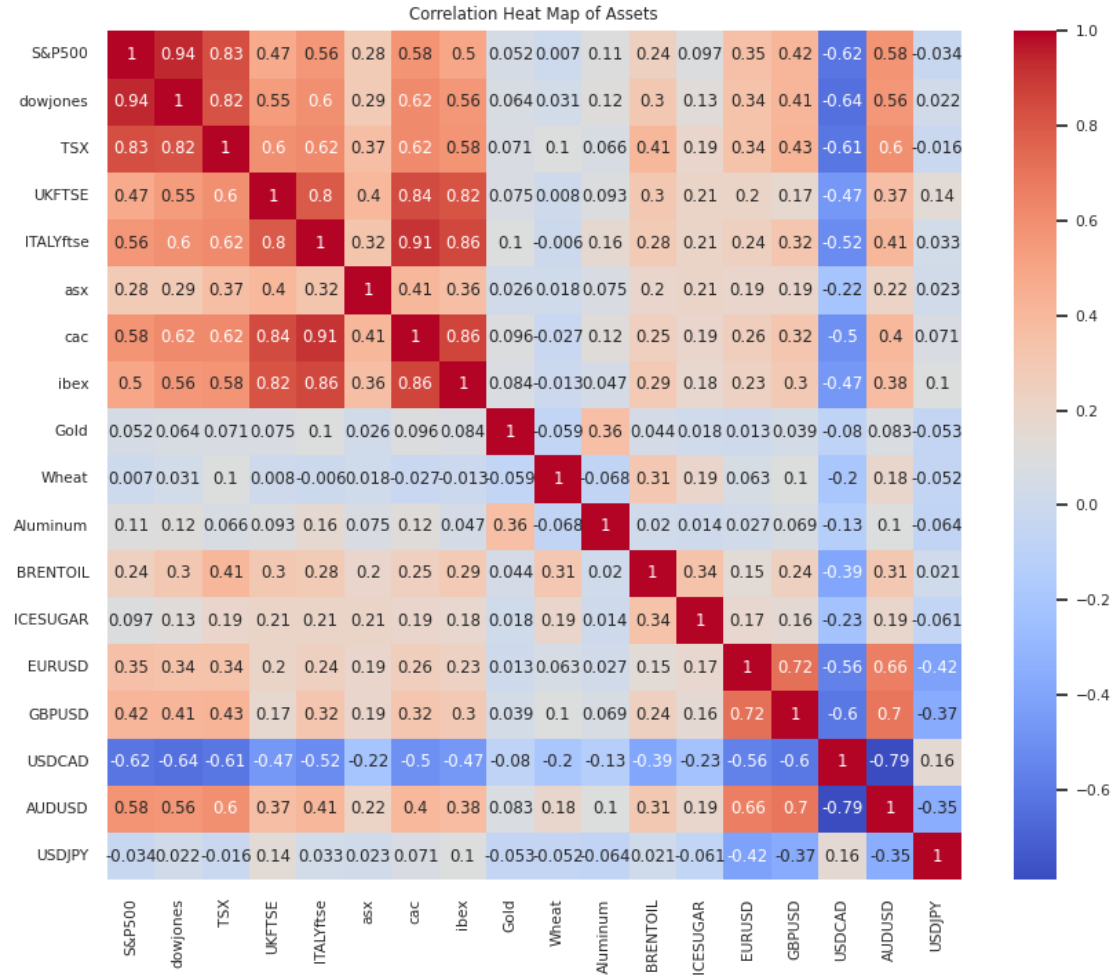
	S&P500	dowjones	TSX	UKFTSE	ITALYftse	asx	cac	ibex	\
S&P500	1.000	0.938	0.833	0.473	0.560	0.285	0.576	0.496	
dowjones	0.938	1.000	0.820	0.545	0.598	0.286	0.616	0.563	
TSX	0.833	0.820	1.000	0.600	0.616	0.368	0.618	0.579	
UKFTSE	0.473	0.545	0.600	1.000	0.796	0.404	0.840	0.816	
ITALYftse	0.560	0.598	0.616	0.796	1.000	0.322	0.910	0.857	
asx	0.285	0.286	0.368	0.404	0.322	1.000	0.406	0.364	
cac	0.576	0.616	0.618	0.840	0.910	0.406	1.000	0.864	
ibex	0.496	0.563	0.579	0.816	0.857	0.364	0.864	1.000	
Gold	0.052	0.064	0.071	0.075	0.104	0.026	0.096	0.084	
Wheat	0.007	0.031	0.101	0.008	-0.006	0.018	-0.027	-0.013	
Aluminum	0.111	0.115	0.066	0.093	0.164	0.075	0.123	0.047	
BRENTOIL	0.238	0.301	0.408	0.305	0.280	0.195	0.246	0.292	
ICESUGAR	0.097	0.132	0.191	0.207	0.209	0.208	0.194	0.179	
EURUSD	0.355	0.340	0.339	0.203	0.244	0.189	0.255	0.230	
GBPUSD	0.416	0.415	0.426	0.171	0.316	0.194	0.316	0.296	
USDCAD	-0.618	-0.639	-0.610	-0.465	-0.522	-0.216	-0.503	-0.465	
AUDUSD	0.577	0.563	0.599	0.366	0.406	0.216	0.403	0.379	
USDJPY	-0.034	0.022	-0.016	0.144	0.033	0.023	0.071	0.103	

	Gold	Wheat	Aluminum	BRENTOIL	ICESUGAR	EURUSD	GBPUSD	USDCAD	\
S&P500	0.052	0.007	0.111	0.238	0.097	0.355	0.416	-0.618	
dowjones	0.064	0.031	0.115	0.301	0.132	0.340	0.415	-0.639	
TSX	0.071	0.101	0.066	0.408	0.191	0.339	0.426	-0.610	
UKFTSE	0.075	0.008	0.093	0.305	0.207	0.203	0.171	-0.465	
ITALYftse	0.104	-0.006	0.164	0.280	0.209	0.244	0.316	-0.522	
asx	0.026	0.018	0.075	0.195	0.208	0.189	0.194	-0.216	
cac	0.096	-0.027	0.123	0.246	0.194	0.255	0.316	-0.503	
ibex	0.084	-0.013	0.047	0.292	0.179	0.230	0.296	-0.465	
Gold	1.000	-0.059	0.363	0.044	0.018	0.013	0.039	-0.080	
Wheat	-0.059	1.000	-0.068	0.310	0.191	0.063	0.103	-0.195	
Aluminum	0.363	-0.068	1.000	0.020	0.014	0.027	0.069	-0.127	
BRENTOIL	0.044	0.310	0.020	1.000	0.339	0.146	0.244	-0.393	
ICESUGAR	0.018	0.191	0.014	0.339	1.000	0.175	0.160	-0.228	
EURUSD	0.013	0.063	0.027	0.146	0.175	1.000	0.719	-0.560	
GBPUSD	0.039	0.103	0.069	0.244	0.160	0.719	1.000	-0.605	
USDCAD	-0.080	-0.195	-0.127	-0.393	-0.228	-0.560	-0.605	1.000	
AUDUSD	0.083	0.185	0.100	0.307	0.187	0.663	0.698	-0.790	
USDJPY	-0.053	-0.052	-0.064	0.021	-0.061	-0.419	-0.366	0.160	

	AUDUSD	USDJPY
S&P500	0.577	-0.034
dowjones	0.563	0.022
TSX	0.599	-0.016
UKFTSE	0.366	0.144
ITALYftse	0.406	0.033
asx	0.216	0.023
cac	0.403	0.071
ibex	0.379	0.103
Gold	0.083	-0.053
Wheat	0.185	-0.052
Aluminum	0.100	-0.064
BRENTOIL	0.307	0.021
ICESUGAR	0.187	-0.061
EURUSD	0.663	-0.419
GBPUSD	0.698	-0.366
USDCAD	-0.790	0.160
AUDUSD	1.000	-0.351
USDJPY	-0.351	1.000

[ ]:
```
# Create a heatmap
plt.figure(figsize=(10, 8))
sns.set(font_scale=0.7)
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heat Map of Assets')
```

[ ]: Text(0.5, 1.0, 'Correlation Heat Map of Assets')



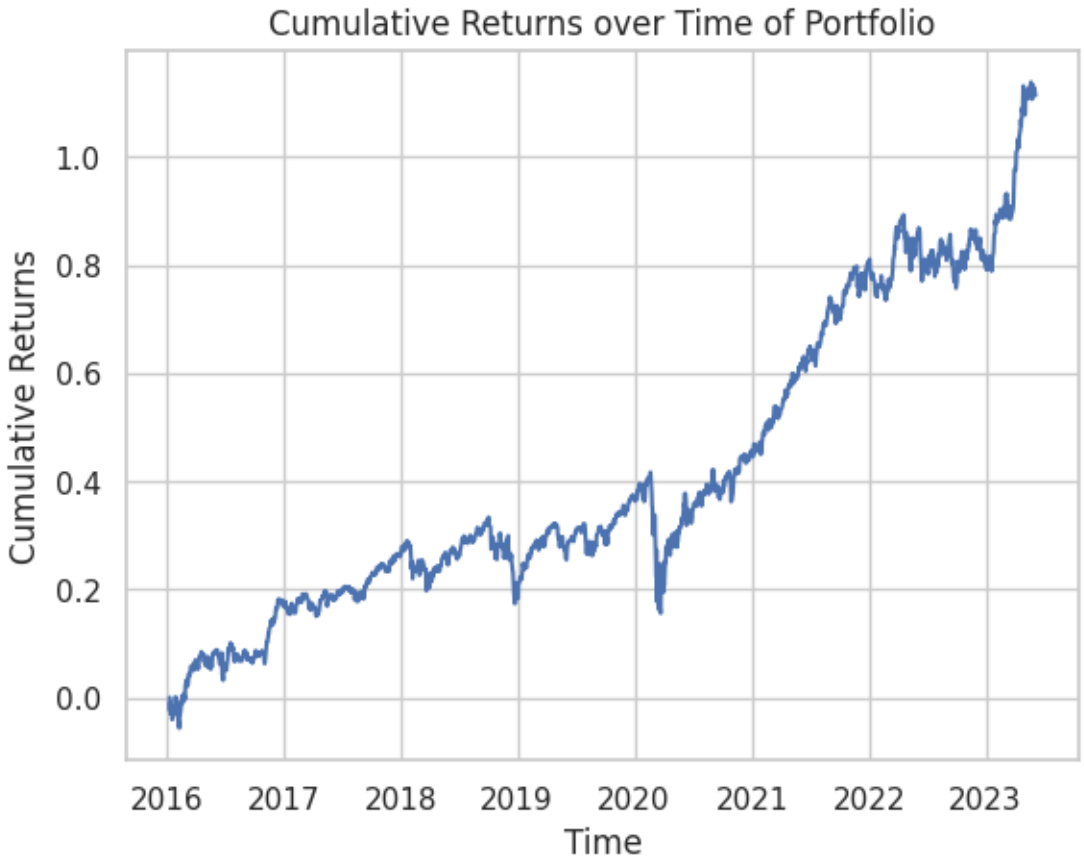
[ ]:
```
import pandas as pd
import numpy as np
import requests
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
from matplotlib.lines import Line2D
from io import BytesIO
from pandas.tseries.offsets import MonthEnd
import pandas_datareader



# URL of your Excel file
```

```
url = 'https://docs.google.com/spreadsheets/d/e/
 ↪2PACX-1vRIXmPmhHHetEyxK-PMz1uyTzBcN2s9-6noA-SdGidWlnqTj1OgzQm7Ekn68D6ASg/pub?
 ↪output=xlsx'

# Send a GET request to the URL
response = requests.get(url)

# Read the content of the response with pandas
data = pd.read_excel(BytesIO(response.content), sheet_name=None)

# Now 'data' is a dictionary where the keys are the names of the sheets and the␣
 ↪values are the dataframes
# You can access the dataframe of a specific sheet like this:
SnP500 = data['S&P500']
dowjones = data['Dow Jones Industrial Average']
TSX = data['TSX']
UKFTSE = data['UK FTSE']
ITALYftse = data['ITALY FTSE MIB']
asx = data['Australia ASX']
cac = data['CAC']
ibex = data['SPAIN IBEX']
Gold = data['Gold']
Coffee = data['Coffee']
Wheat = data['Wheat']
Aluminum = data['Aluminum']
BRENTOIL = data['BRENTOIL']
ICESUGAR = data['ICESUGAR']
EURUSD = data['EURUSD']
GBPUSD = data['GBPUSD']
USDCAD = data['USDCAD']
AUDUSD = data['AUDUSD']
USDJPY = data['USDJPY']
def analyze_dataframe(df1, df_name):
    df=df1.copy()
    df.dropna()
    # Ensure 'Date' column is in datetime format
    df['Date'] = pd.to_datetime(df['Date'])
    df.set_index('Date', inplace=True)
    df = df.sort_values(by='Date')
    # Calculate daily returns
    df['Daily Returns'] = df['Price'].pct_change()

    # Define momentum signal function and apply it
    def momentum_signal(data, lookback_period):
        return np.sign(data.shift(lookback_period).mean())
```

```
    lookback_periods = [21, 63, 252]  # Approximate 1-month, 3-months, and␣
↪12-months in trading days
    for period in lookback_periods:
        df[f'{period}D Momentum Signal'] = momentum_signal(df['Daily Returns'],␣
↪period)

    # Average the signals
    df['Momentum Signal'] = df[[f'{period}D Momentum Signal' for period in␣
↪lookback_periods]].mean(axis=1)

    # Compute strategy returns without volatility scaling
    df['Momentum Returns'] = df['Momentum Signal'].shift() * df['Daily Returns']

    # Transaction cost and fees
    df['Transaction Costs'] = 0.0000  # This is a placeholder.
    df['Fees'] = 0.0000 + 000.000 * np.maximum(df['Momentum Returns'] -␣
↪df['Transaction Costs'], 0)
    df['Net Momentum Returns'] = df['Momentum Returns'] - df['Transaction␣
↪Costs'] - df['Fees']

    return df['Net Momentum Returns']

products = ['S&P500', 'ICESUGAR', 'USDJPY']

# Create an empty DataFrame to hold the returns of all assets
portfolio_returns = pd.DataFrame()

for i in products:
  portfolio_returns[i] = analyze_dataframe(data[i], i)

# Calculate the portfolio returns as the mean of the asset returns
portfolio_returns['Portfolio'] = portfolio_returns.mean(axis=1)

# Calculate the portfolio's annualized return, volatility, and Sharpe ratio
portfolio_returns['Annualized Returns'] = portfolio_returns['Portfolio'].mean()␣
↪* 252
portfolio_returns['Annualized Volatility'] = portfolio_returns['Portfolio'].
↪std() * np.sqrt(252)
portfolio_returns['Sharpe Ratio'] = portfolio_returns['Annualized Returns'] /␣
↪portfolio_returns['Annualized Volatility']

# Print the results
print(f"Portfolio Annualized Excess Return: {portfolio_returns['Annualized␣
↪Returns'].iloc[-1]}")
print(f"Portfolio Volatility: {portfolio_returns['Annualized Volatility'].
↪iloc[-1]}")
```

```
print(f"Portfolio Sharpe Ratio: {portfolio_returns['Sharpe Ratio'].iloc[-1]}")

# Plot the cumulative returns
cumulative_returns = (1 + portfolio_returns['Portfolio'].fillna(0)).cumprod() -␣
↪1
plt.plot(cumulative_returns)
plt.xlabel('Time')
plt.ylabel('Cumulative Returns')
plt.title('Cumulative Returns over Time of Portfolio')
plt.show()
```

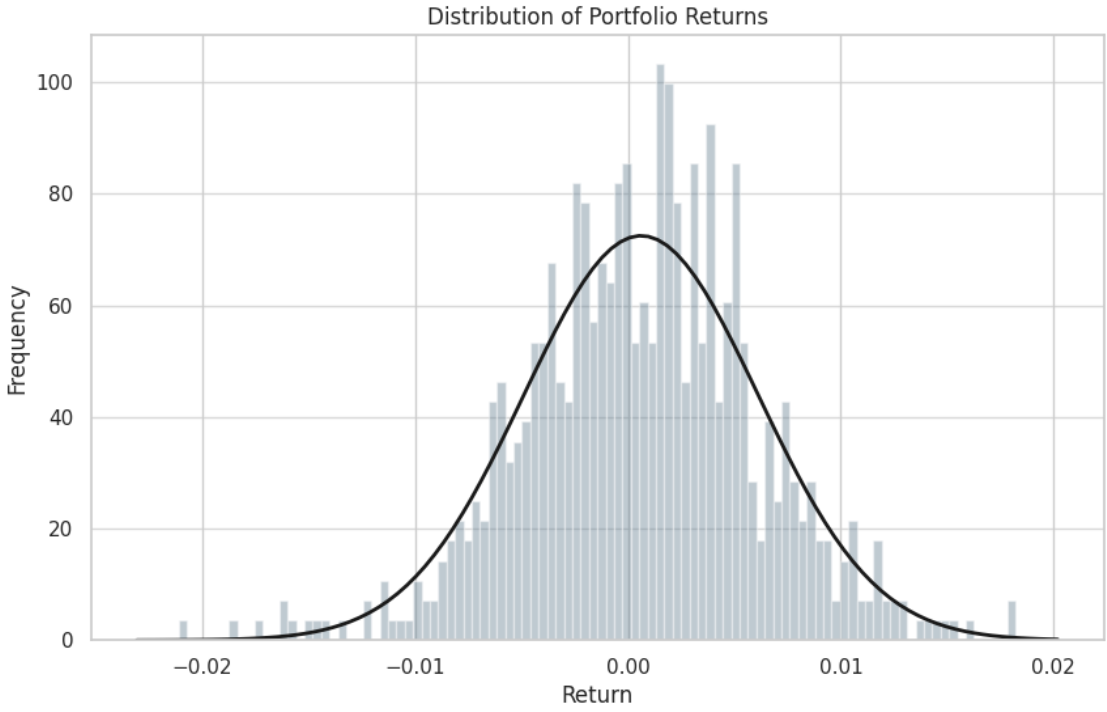
```
Portfolio Annualized Excess Return: 0.10752087462653226
Portfolio Volatility: 0.11113749201978658
Portfolio Sharpe Ratio: 0.9674581698081649
```



[ ]:
```
# Drop NA values
portfolio_returns = portfolio_returns.dropna()

# Compute the mean and standard deviation of the portfolio returns
```

```
mu = portfolio_returns['Portfolio'].mean()
sigma = portfolio_returns['Portfolio'].std()

# Set the figure size
plt.figure(figsize=(10,6))

# Create a seaborn histogram with a density curve
sns.distplot(portfolio_returns['Portfolio'], bins=100, kde=False,␣
 ↪norm_hist=True, color='#607c8e')

# Plot the standard normal distribution
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mu, sigma)
plt.plot(x, p, 'k', linewidth=2)

plt.grid(axis='y', alpha=0.75)
plt.xlabel('Return')
plt.ylabel('Frequency')
plt.title('Distribution of Portfolio Returns')

plt.show()
```

```
<ipython-input-50-c30bee9df73e>:12: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(portfolio_returns['Portfolio'], bins=100, kde=False,
norm_hist=True, color='#607c8e')
```



[ ]:
```
# URL of your Excel file
url = 'https://docs.google.com/spreadsheets/d/e/
 ↪2PACX-1vQtEc8fjeQ8kkWNYn7_wcBAr8WVQNvH29rjyj6Vk20F53VZEv8OWwd_7FW6WCXRpczXDRVwNMGv4cRI/
 ↪pub?output=xlsx'

# Send a GET request to the URL
response = requests.get(url)

# Read the content of the response with pandas
data = pd.read_excel(BytesIO(response.content), sheet_name=None)

# Now 'data' is a dictionary where the keys are the names of the sheets and the␣
 ↪values are the dataframes
# You can access the dataframe of a specific sheet like this:
FF = data['F-F_Research_Data_Factorss.CSV']



FF = pandas_datareader.famafrench.FamaFrenchReader('F-F_Research_Data_Factors',␣
 ↪start='01-2016', end='12-2022')
FF = FF.read()[0]/100
FF.columns = 'MktRF','SMB','HML','RF'
FF['Mkt']  = FF['MktRF'] + FF['RF']
```

```
FF  = FF.reset_index().rename(columns = {"Date" : "date"}).copy()
FF['date'] = pd.DataFrame(FF[['date']].values.astype('datetime64[ns]')) +␣
 ↪MonthEnd(0)
FF['Year'] = FF['date'].dt.year
FF['Month'] = FF['date'].dt.month
FF['Cumulative Market Return'] = (1 + FF['Mkt']).cumprod() - 1
FF = FF[['Year','Month','RF','Mkt','Cumulative Market Return']]

# Assuming your DataFrame is named df
FF['Date'] = pd.to_datetime(FF[['Year', 'Month']].assign(DAY=1))

plt.figure(figsize=(10,6))
plt.plot(FF['Date'], FF['Cumulative Market Return'])
plt.xlabel('Date')
plt.ylabel('Cumulative Market Return')
plt.title('Cumulative Market Return')
plt.show()
ann_mean = FF['Mkt'].mean() * 12
ann_std = FF['Mkt'].std() * np.sqrt(12)
ann_sr = (ann_mean - FF['RF'].mean()*12) / ann_std

ann_sr
```

[ ]: 0.6565733748070015

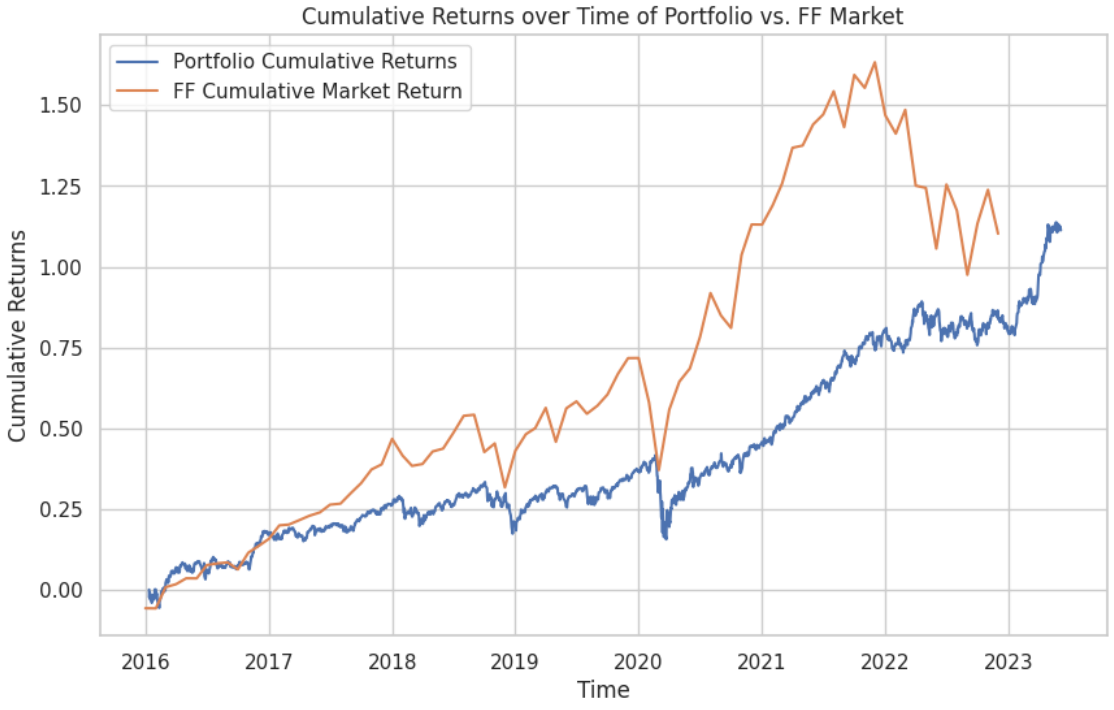
[ ]:
```
ann_mean
```

[ ]: 0.12087142857142859

[ ]:
```
ann_std
```

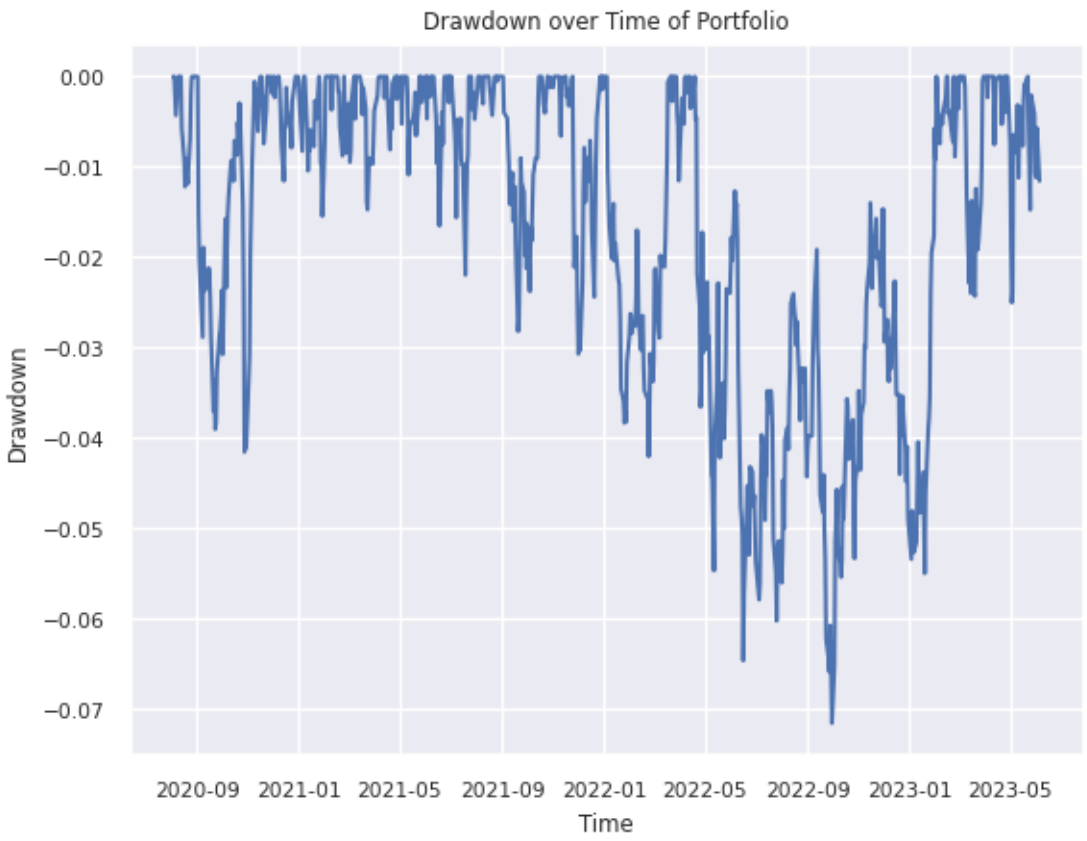
[ ]: 0.1692336149740272

[ ]:
```
# Calculate the cumulative returns
cumulative_returns = (1 + portfolio_returns['Portfolio'].fillna(0)).cumprod() -␣
↪1

# Plot the cumulative returns
plt.figure(figsize=(10,6))
plt.plot(cumulative_returns, label='Portfolio Cumulative Returns')

FF['Date'] = pd.to_datetime(FF[['Year', 'Month']].assign(DAY=1))

# Plot the cumulative market return
plt.plot(FF['Date'], FF['Cumulative Market Return'], label='FF Cumulative␣
↪Market Return')

plt.xlabel('Time')
plt.ylabel('Cumulative Returns')
plt.title('Cumulative Returns over Time of Portfolio vs. FF Market')
plt.legend()
plt.show()
```



[ ]:
```python
def calculate_drawdown(returns):

    cumulative_returns = (1 + returns).cumprod()
    previous_peaks = cumulative_returns.cummax()
    drawdowns = (cumulative_returns - previous_peaks) / previous_peaks

    return drawdowns


# Calculate the drawdown for the portfolio returns
portfolio_drawdowns = calculate_drawdown(portfolio_returns['Portfolio'].
 ↪fillna(0))

# Calculate the maximum drawdown
max_drawdown = portfolio_drawdowns.min()

# Print the maximum drawdown
print(f"Maximum Drawdown: {max_drawdown}")

# Plot the drawdown over time
plt.plot(portfolio_drawdowns)
plt.xlabel('Time')
plt.ylabel('Drawdown')
plt.title('Drawdown over Time of Portfolio')
plt.show()
```

```
Maximum Drawdown: -0.0716091565446669
```


Drawdown over Time of Portfolio
0.00
−0.01
−0.02
−0.03
−0.04
−0.05
−0.06
−0.07
Drawdown
2020-09
2021-01
2021-05
2021-09
2022-01
2022-05
2022-09
2023-01
2023-05
Time

[ ]: