SSGA FICC Research Analyst Assignment

Rayavarapu Laxman Rao
CH23M028

Part 1:

The objective of this analysis is to identify and analyze trends in the stock prices of 11 assets over a given period. Trends are essential for understanding market behavior and making informed investment decisions. This analysis leverages Python programming to process data efficiently and generate meaningful insights through visualizations.

Exponential Moving Average (EMA):
The EMA is a type of weighted moving average where more recent prices are given greater weight. This makes it more sensitive to recent price changes compared to a simple moving average (SMA), which gives equal weight to all prices in the period.

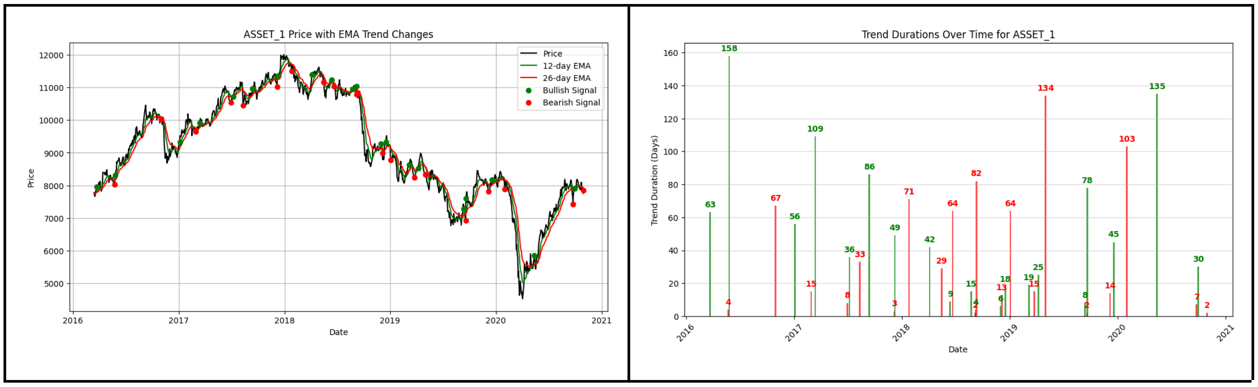
12-period EMA (Fast EMA): This is a short-term moving average, highly sensitive to recent price movements. It reacts quickly to price changes.

26-period EMA (Slow EMA): This is a longer-term moving average. It responds slower to price changes and helps identify broader market trends.

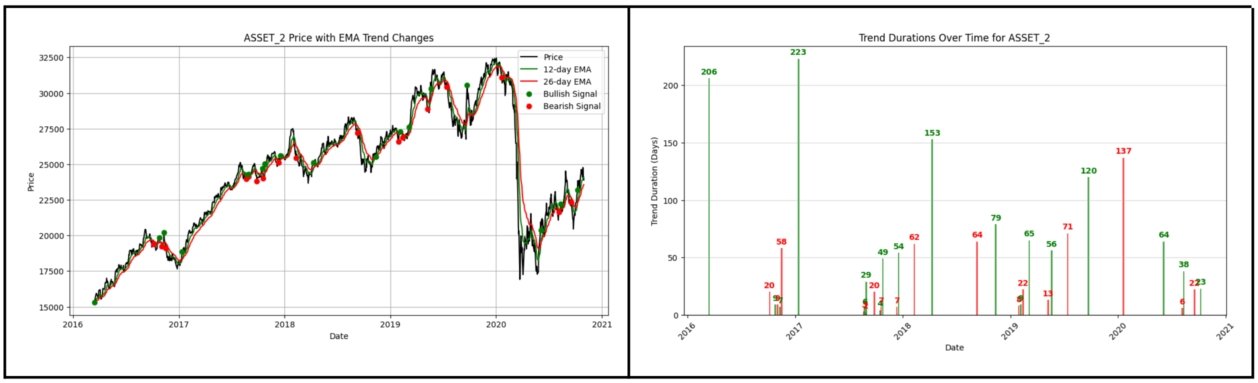
A bullish trend is identified when the 12-period EMA crosses above the 26-period EMA. This crossover indicates that recent prices are rising faster than the longer-term average, signaling an upward momentum. A bearish trend occurs when the 12-period EMA crosses below the 26-period EMA. This indicates that recent prices are declining faster than the longer-term average, signaling downward momentum.

Uptrend Duration: The time between the point where the 12 EMA crosses above the 26 EMA and the next point where the 12 EMA crosses below the 26 EMA.

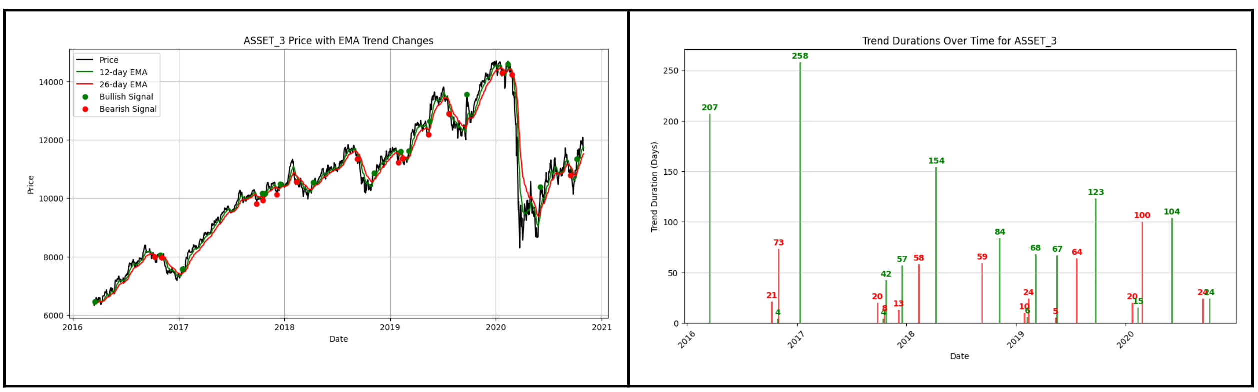
Downtrend Duration: The time between the point where the 12 EMA crosses below the 26 EMA and the next point where the 12 EMA crosses above the 26 EMA.



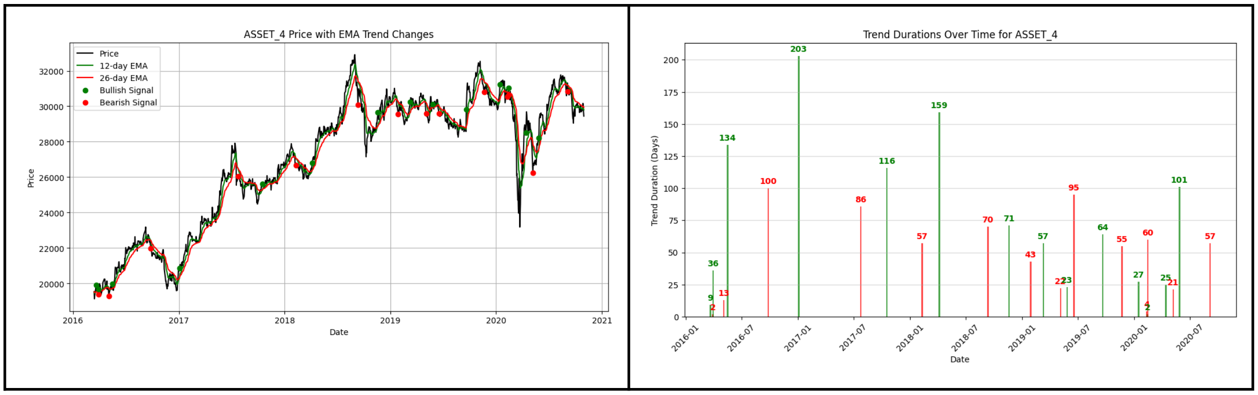
Trend analysis of ASSET_1



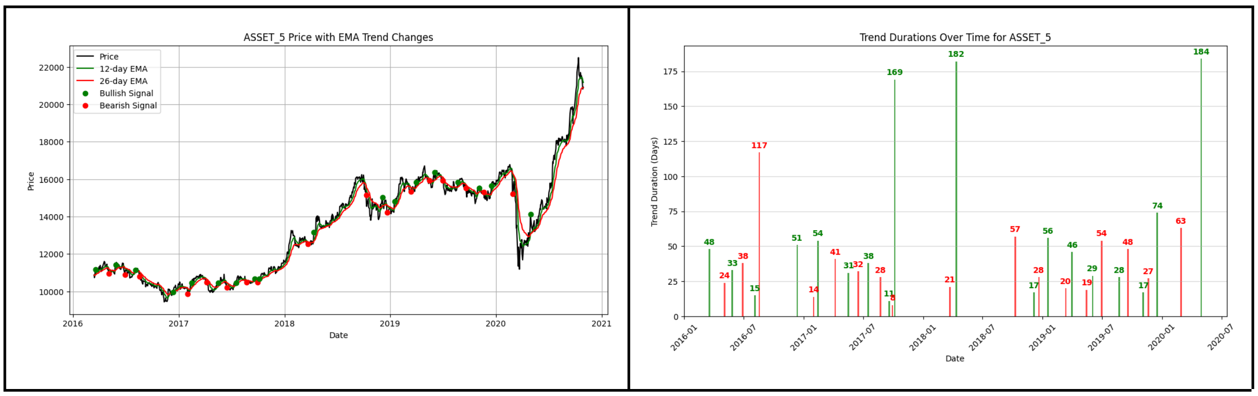
Trend analysis of ASSET_2



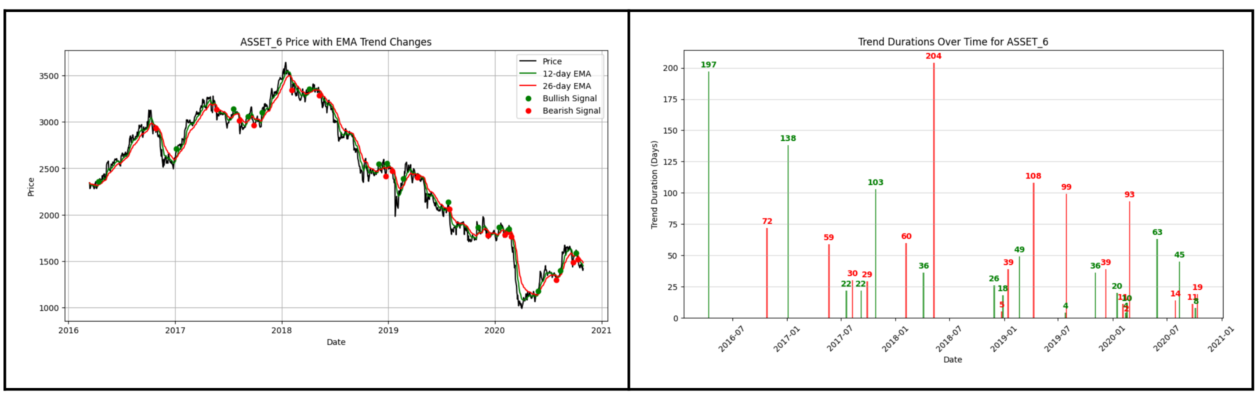
Trend analysis of ASSET_3



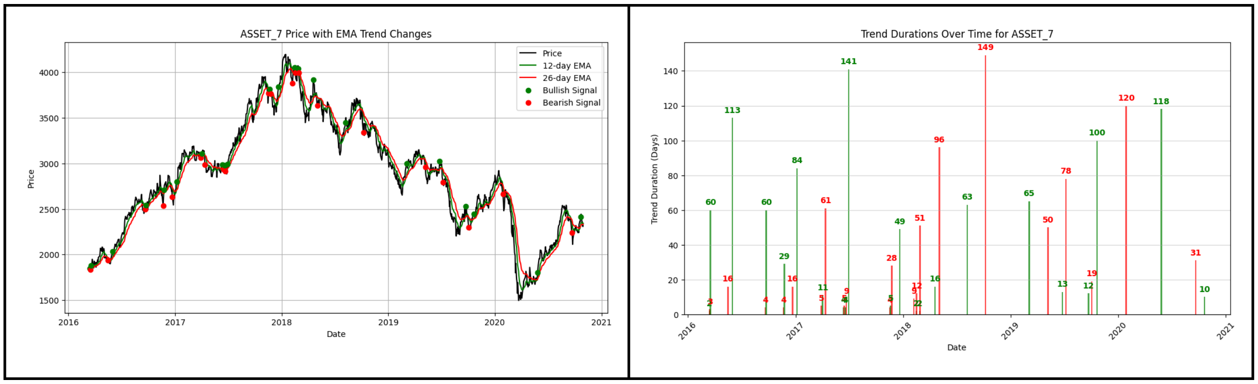
Trend analysis of ASSET_4



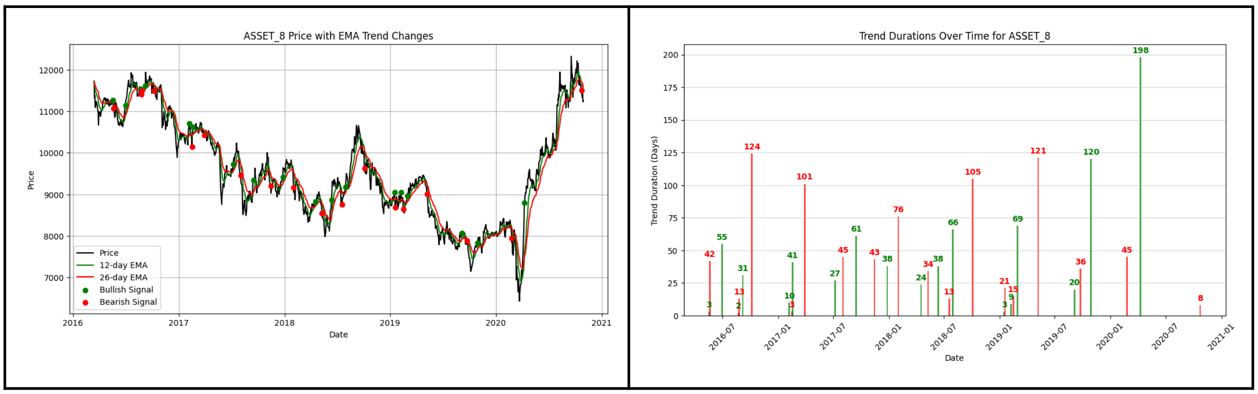
Trend analysis of ASSET_5



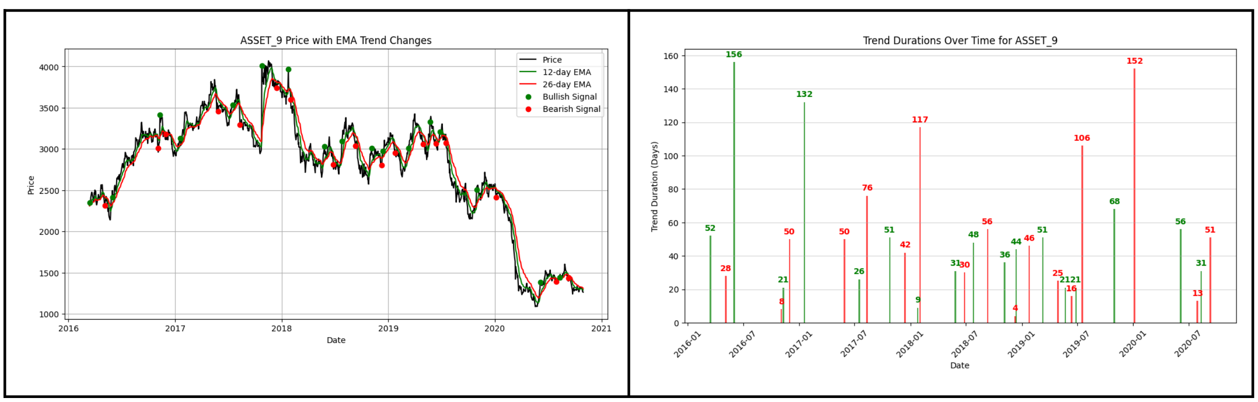
Trend analysis of ASSET_6



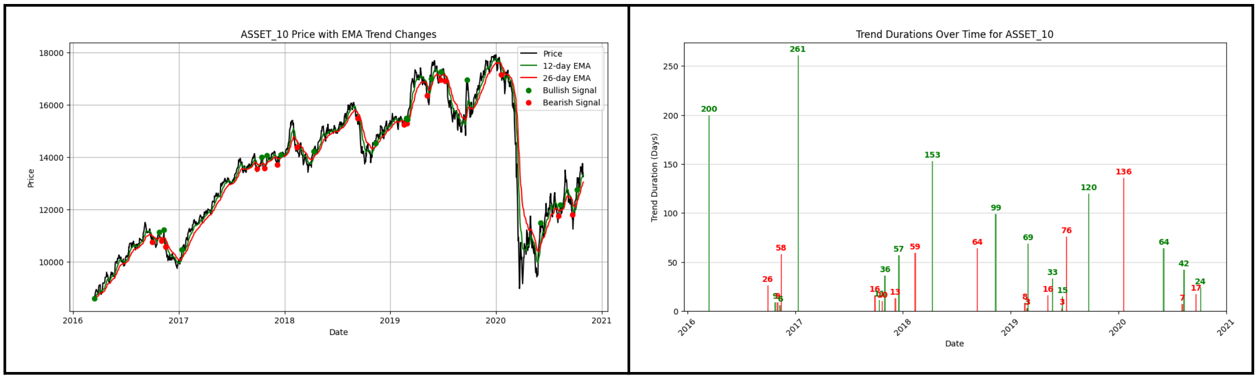
Trend analysis of ASSET_7



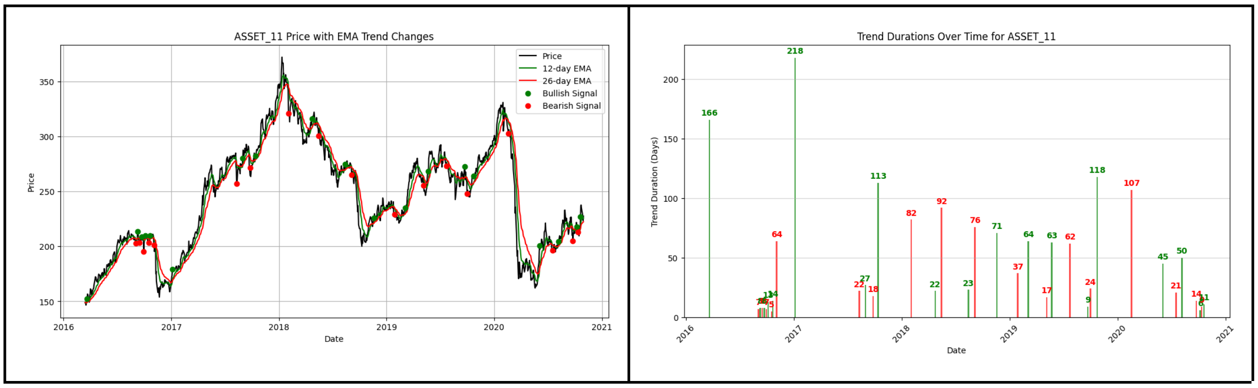
Trend analysis of ASSET_8



Trend analysis of ASSET_9



Trend analysis of ASSET_10



Trend analysis of ASSET_11

Code for the part 1:

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

# Load data
df = pd.read_csv(r"C:\Users\hi\Downloads\ALLDATA.csv")
df['Dates'] = pd.to_datetime(df['Dates'], format='%d-%m-%Y')
df.set_index('Dates', inplace=True)

# Define EMA parameters
short_window = 12
long_window = 26

# Store trend details
trend_summary = {}

for asset in df.columns:
```

```
    asset_data = df[asset].copy()

    # Compute Exponential Moving Averages (EMAs)
    asset_data_short = asset_data.ewm(span=short_window,
adjust=False).mean()
    asset_data_long = asset_data.ewm(span=long_window,
adjust=False).mean()

    # Create signal: 1 for uptrend, -1 for downtrend
    signal = np.where(asset_data_short > asset_data_long, 1, -1)

    # Identify trend changes
    trend_changes = np.diff(signal)
    trend_start_indices = np.where(np.abs(trend_changes) == 2)[0] + 1  #
+1 to shift to correct index

    # Collect trend data
    trends = []
    for i in range(len(trend_start_indices)):
        start_idx = trend_start_indices[i]
        end_idx = trend_start_indices[i+1] if i < len(trend_start_indices)
- 1 else len(asset_data) - 1

        trend_type = 'Uptrend' if signal[start_idx] == 1 else 'Downtrend'
        duration = (asset_data.index[end_idx] -
asset_data.index[start_idx]).days + 1

        trends.append({
            'Start Date': asset_data.index[start_idx],
            'End Date': asset_data.index[end_idx],
            'Trend Type': trend_type,
            'Duration (days)': duration
        })

    trend_summary[asset] = {'Trend Count': len(trends), 'Trends': trends}

    # --- PRICE CHART WITH TREND LINES ---
    plt.figure(figsize=(12, 6))
    plt.plot(asset_data.index, asset_data, label='Price', color='black')
```

```
    plt.plot(asset_data.index, asset_data_short,
label=f'{short_window}-day EMA', alpha=1, color='green')
    plt.plot(asset_data.index, asset_data_long, label=f'{long_window}-day
EMA', alpha=1, color='red')

    # Flags to control legend entry appearance
    bullish_plotted = False
    bearish_plotted = False

    # Plot trend markers
    for i in range(len(trend_start_indices)):
        start_idx = trend_start_indices[i]
        if signal[start_idx] == 1:  # Bullish
            plt.scatter(asset_data.index[start_idx],
asset_data.iloc[start_idx],
                        color='green', zorder=5, label='Bullish Signal' if
not bullish_plotted else "")
            bullish_plotted = True  # Ensure legend appears only once
        else:  # Bearish
            plt.scatter(asset_data.index[start_idx],
asset_data.iloc[start_idx],
                        color='red', zorder=5, label='Bearish Signal' if
not bearish_plotted else "")
            bearish_plotted = True  # Ensure legend appears only once

    plt.title(f'{asset} Price with EMA Trend Changes')
    plt.xlabel('Date')
    plt.ylabel('Price')
    plt.legend()
    plt.grid(True)
    plt.show()

    # --- HISTOGRAM PLOTTING ---
    trend_dates = [t['Start Date'] for t in trends]
    trend_durations = [t['Duration (days)'] for t in trends]
    trend_colors = ['green' if t['Trend Type'] == 'Uptrend' else 'red' for
t in trends]

    plt.figure(figsize=(12, 6))
```

```
    plt.bar(trend_dates, trend_durations, color=trend_colors, alpha=0.7,
width=5)

    # Add duration labels on top of bars
    for i in range(len(trend_dates)):
        plt.text(trend_dates[i], trend_durations[i] + 2,
str(trend_durations[i]),
                  ha='center', va='bottom', fontsize=10, fontweight='bold',
color=trend_colors[i])

    plt.xlabel("Date")
    plt.ylabel("Trend Duration (Days)")
    plt.title(f"Trend Durations Over Time for {asset}")

    # Grid for clarity
    plt.grid(axis='y', alpha=0.5)
    plt.xticks(rotation=45)
    plt.show()

# Display trend summary
for asset, info in trend_summary.items():
    print(f"\n{asset} - Number of Trends Detected: {info['Trend Count']}")
    print(pd.DataFrame(info['Trends']))
```

Part 2:

Shortcomings:

False Signals: The 12 and 26-period EMAs may generate false signals during sideways or choppy markets. In such cases, the price could fluctuate between the EMAs, leading to multiple buy/sell signals that do not yield profitable trades.

Short-Term Focus: The 12 and 26-period EMAs are relatively short-term, and they may fail to capture longer-term trends. During periods of strong momentum, the strategy might exit positions prematurely.

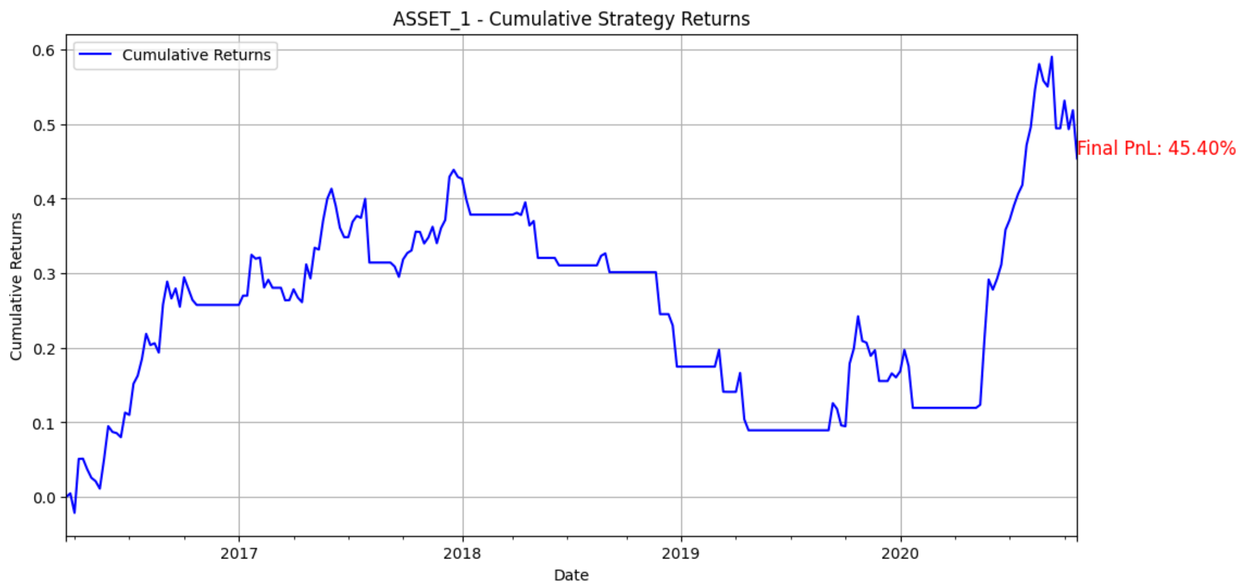
Lack of Risk Management: The strategy assumes that all trades are equally weighted and that there is no risk management in place (e.g., no stop losses, no portfolio diversification). In real trading, such an approach can lead to significant losses if the market moves unexpectedly.

Improvements to the Strategy:

Volume: Volume can be used as a confirming indicator. If an asset is trending up with increasing volume, it might indicate a more reliable trend. Similarly, if the trend is accompanied by decreasing volume, it could signal weakening momentum.

Risk Management: Implementing stop-loss orders, position sizing, or a risk-reward ratio could help mitigate losses and improve the overall risk-return profile of the strategy.

Alternative Trend Indicators: MACD is often considered one of the most widely used and reliable trend indicators in technical analysis, and it can provide valuable insights into both trend direction and momentum.

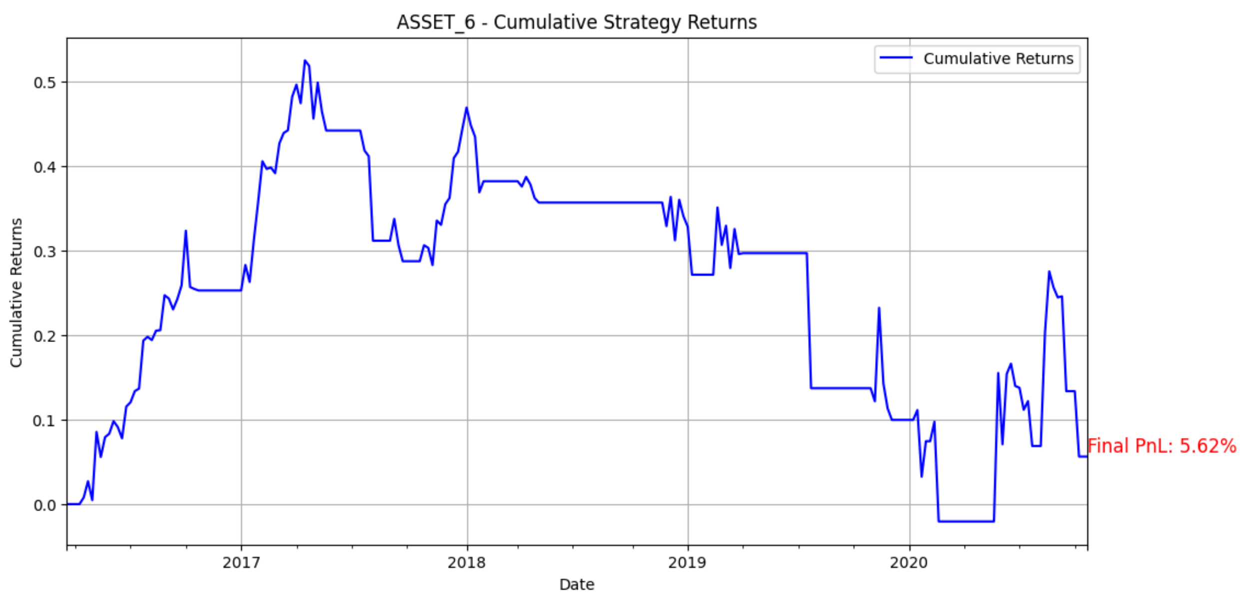

ASSET_1 - Cumulative Strategy Returns
Cumulative Returns
Final PnL: 45.40%
Cumulative Returns
Date
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2017
2018
2019
2020

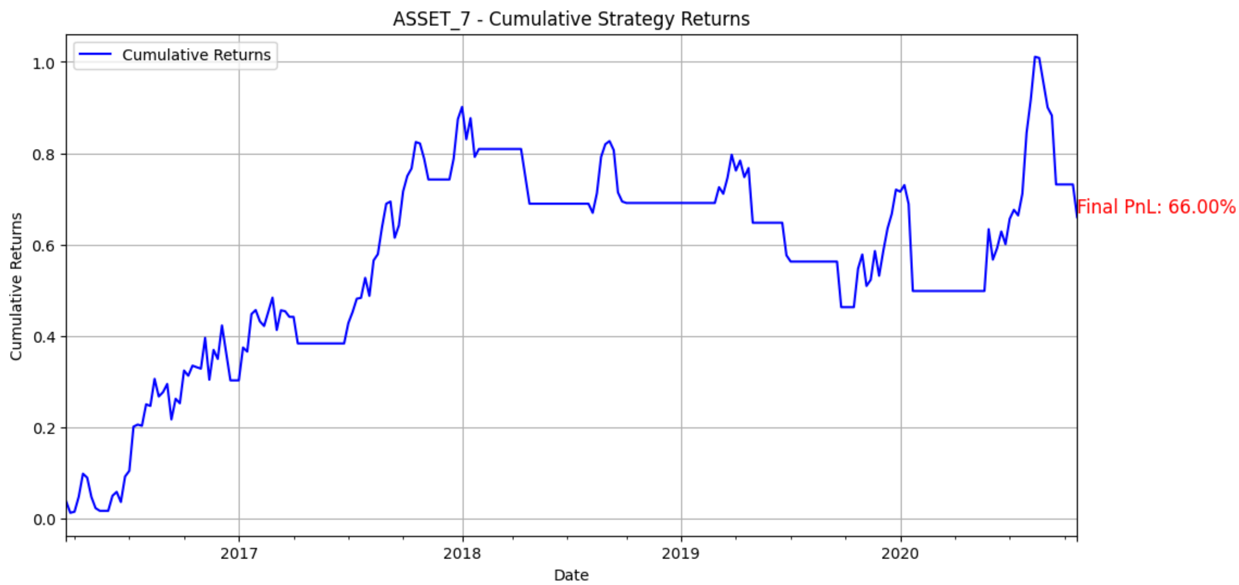

ASSET_2 - Cumulative Strategy Returns
Cumulative Returns
Final PnL: 80.31%
Cumulative Returns
Date
1.0
0.8
0.6
0.4
0.2
0.0
2017
2018
2019
2020

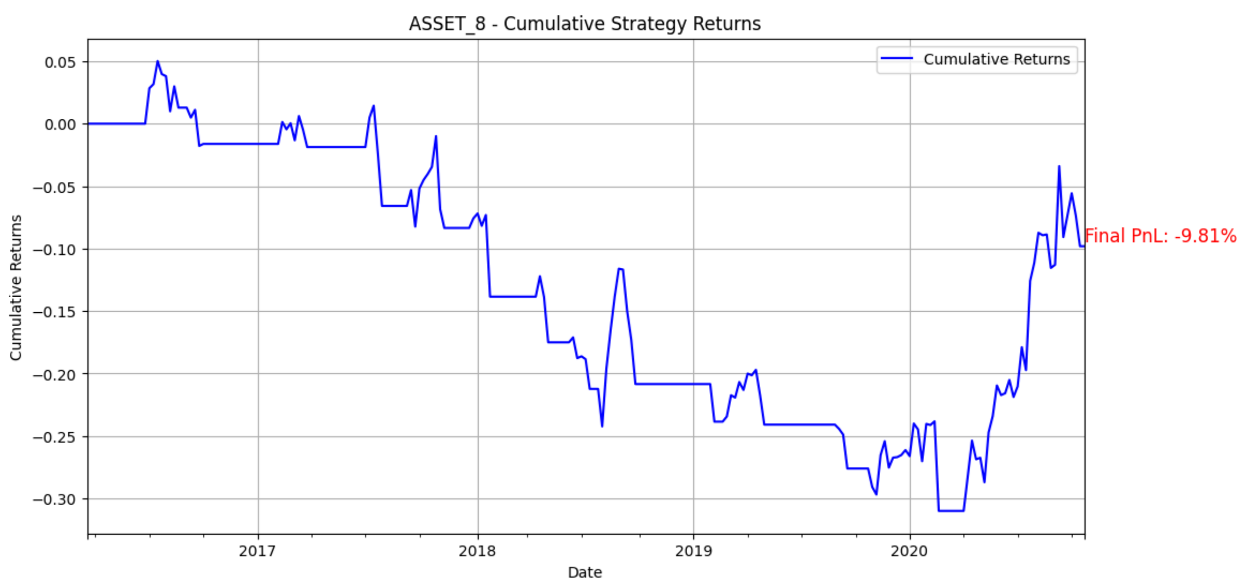

ASSET_3 - Cumulative Strategy Returns
Cumulative Returns
Final PnL: 102.67%
Cumulative Returns
Date
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2017
2018
2019
2020


ASSET_4 - Cumulative Strategy Returns
Cumulative Returns
Final PnL: 58.64%
Cumulative Returns
Date
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2017
2018
2019
2020


ASSET_5 - Cumulative Strategy Returns
Cumulative Returns
Final PnL: 43.04%
Cumulative Returns
Date
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
2017
2018
2019
2020


ASSET_6 - Cumulative Strategy Returns
Cumulative Returns
Final PnL: 5.62%
Cumulative Returns
Date
0.5
0.4
0.3
0.2
0.1
0.0
2017
2018
2019
2020


ASSET_7 - Cumulative Strategy Returns
Cumulative Returns
Cumulative Returns
Date
Final PnL: 66.00%
1.0
0.8
0.6
0.4
0.2
0.0
2017
2018
2019
2020


ASSET_8 - Cumulative Strategy Returns
Cumulative Returns
Cumulative Returns
Date
Final PnL: -9.81%
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.25
−0.30
2017
2018
2019
2020


ASSET_9 - Cumulative Strategy Returns
Cumulative Returns
Cumulative Returns
0.4
0.3
0.2
0.1
0.0
−0.1
2017
2018
2019
2020
Date
Final PnL: -7.87%


ASSET_10 - Cumulative Strategy Returns
Cumulative Returns
Cumulative Returns
1.0
0.8
0.6
0.4
0.2
0.0
2017
2018
2019
2020
Date
Final PnL: 71.41%

Code for part 2:

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

# Load data
df = pd.read_csv(r"C:\Users\hi\Downloads\ALLDATA.csv")
df['Dates'] = pd.to_datetime(df['Dates'], format='%d-%m-%Y')
df.set_index('Dates', inplace=True)

# Define parameters for EMA
short_window = 12
long_window = 26

for asset in df.columns:
    asset_data = df[asset].copy()

    # Compute EMAs
    short_ema = asset_data.ewm(span=short_window, adjust=False).mean()
    long_ema = asset_data.ewm(span=long_window, adjust=False).mean()

    # Resample EMAs to weekly (last value of each week)
    short_ema_weekly = short_ema.resample('W').last()
    long_ema_weekly = long_ema.resample('W').last()
```

```python
    # Generate weekly signal (1 if uptrend, 0 otherwise)
    weekly_signal = (short_ema_weekly > long_ema_weekly).astype(int)

    # Weekly prices and forward returns
    weekly_prices = asset_data.resample('W').last()
    forward_returns = (weekly_prices.shift(-1) / weekly_prices) - 1

    # Strategy returns
    strategy_returns = weekly_signal * forward_returns
    strategy_returns.dropna(inplace=True)  # Drop last NaN

    # Cumulative returns
    cumulative_returns = (1 + strategy_returns).cumprod() - 1

    # Plot cumulative returns
    plt.figure(figsize=(12, 6))
    cumulative_returns.plot(label='Cumulative Returns', color='blue')

    # Add final cumulative PnL as annotation
    final_return = cumulative_returns.iloc[-1] if not
cumulative_returns.empty else 0
    plt.text(cumulative_returns.index[-1], final_return,
             f'Final PnL: {final_return:.2%}',
             horizontalalignment='left',
             verticalalignment='bottom',
             fontsize=12, color='red')

    plt.title(f'{asset} - Cumulative Strategy Returns')
    plt.xlabel('Date')
    plt.ylabel('Cumulative Returns')
    plt.legend()
    plt.grid(True)
    plt.show()

    # Print final cumulative return
    print(f"{asset} Final Cumulative Return: {final_return:.2%}\n")
```

Part 3:

1. Introduction:

MPT helps investors pick a mix of assets that balances the amount of risk they want to take with the returns they expect. The idea is that by combining different investments, you can minimize risk and maximize returns at the same time.

Minimum Variance Portfolio (MVP):

This portfolio focuses on minimizing risk. The goal is to find the combination of assets (stocks, bonds, etc.) that results in the lowest possible risk for a given set of assets. The MVP is ideal for investors who want to avoid big fluctuations in their investment value. It's a conservative approach, prioritizing stability.

Tangency Portfolio:

The Tangency Portfolio seeks to balance risk and return in the most efficient way. It's found by combining the riskiest assets with the highest expected return, such that the portfolio touches the efficient frontier at the point where the risk-adjusted return is maximized. This portfolio gives investors the best return for the level of risk they are willing to accept. It's considered the ideal portfolio for those who want to take on more risk to get higher returns.

Data Collection and Returns Calculation:
Historical stock prices are imported and converted into daily logarithmic returns, computed as:

$$r_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

Where P_t is the price at time 't'. Daily returns are annualized by scaling mean returns and covariance by 252 trading days, yielding:

$$\mu_{annual} = \mu_{daily} * 252 \text{ and } \Sigma_{annual} = \Sigma_{daily} * 252$$

The expected return of a portfolio with weights w is:

$$\mu_p = w^T \mu$$

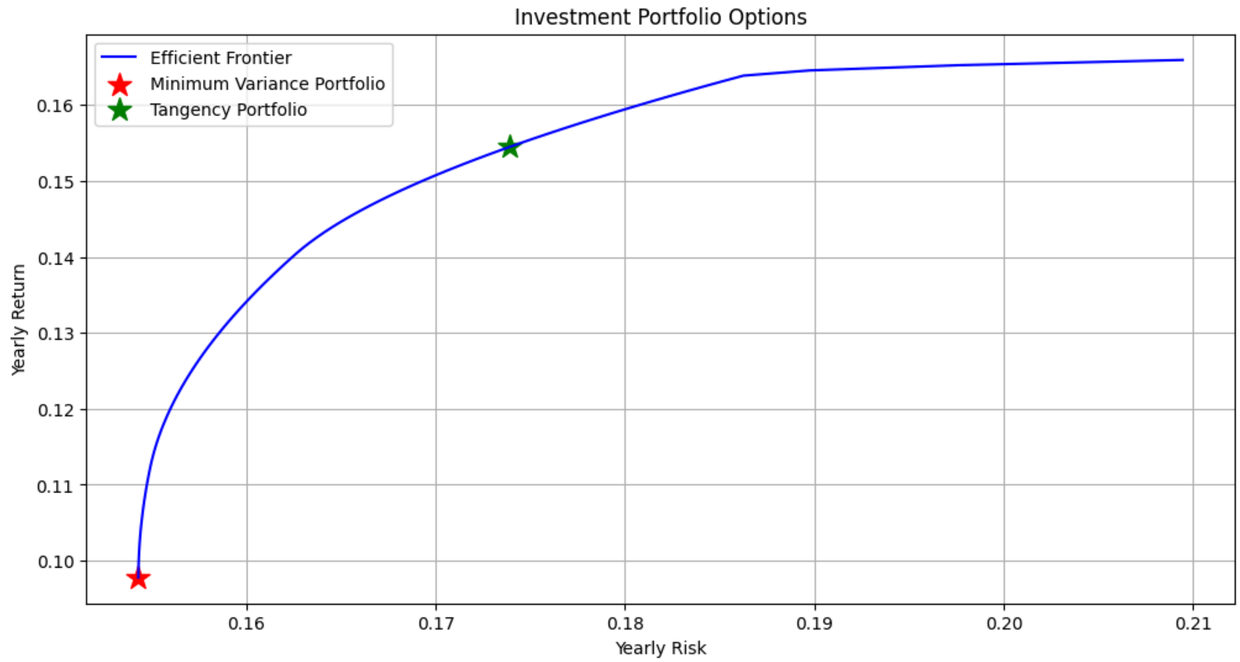
Portfolio risk (volatility) is quantified as:

$$\sigma_p = \sqrt{w^T \Sigma w}$$

The risk-adjusted return metric is defined as:

$$sharp\ ratio = \frac{\mu_p - r_f}{\sigma_p}$$

where r_f is the risk-free rate (assumed 0 here).



Assets	Minimum Variance Portfolio	Tangency Portfolio
ASSET_1	0%	0%
ASSET_2	0%	0%
ASSET_3	1.8%	28.7%
ASSET_4	43.9%	16.6%
ASSET_5	29.1%	54.7%
ASSET_6	4.7%	0%
ASSET_7	0%	0%
ASSET_8	20.5%	0%
ASSET_9	0%	0%
ASSET_10	0%	0%
ASSET_11	0%	0%

Code for part 3:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

# Read stock data from file
df = pd.read_csv(r"C:\Users\hi\Downloads\ALLDATA.csv")

# Convert dates to proper format and set as index
df['Dates'] = pd.to_datetime(df['Dates'], format='%d-%m-%Y')
df.set_index('Dates', inplace=True)

# Calculate daily percentage changes (returns) and remove empty rows
daily_returns = df.pct_change().dropna()

# Convert daily returns to yearly estimates
yearly_returns = daily_returns.mean() * 252          # Average returns per
year
yearly_covariance = daily_returns.cov() * 252        # How stocks move
together per year
num_stocks = len(yearly_returns)

def get_portfolio_return(weights):
    return np.dot(weights, yearly_returns)

def get_portfolio_risk(weights):
    return np.sqrt(weights.T @ yearly_covariance @ weights)

# All weights should add to 100% and no negative investments
constraints = ({'type': 'eq', 'fun': lambda w: np.sum(w) - 1})
weight_limits = [(0, 1) for _ in range(num_stocks)]
start_guess = np.ones(num_stocks)/num_stocks  # Start with equal weights

safest = minimize(get_portfolio_risk,
                  start_guess,
                  method='SLSQP',
                  bounds=weight_limits,
                  constraints=constraints)
```

```python
safe_weights = safest.x
safe_return = get_portfolio_return(safe_weights)
safe_risk = get_portfolio_risk(safe_weights)

def sharpe_ratio(weights):
    return get_portfolio_return(weights) / get_portfolio_risk(weights)

best_ratio = minimize(lambda w: -sharpe_ratio(w),  # We use negative to find maximum
                     start_guess,
                     method='SLSQP',
                     bounds=weight_limits,
                     constraints=constraints)

best_weights = best_ratio.x
best_return = get_portfolio_return(best_weights)
best_risk = get_portfolio_risk(best_weights)


possible_returns = np.linspace(safe_return, yearly_returns.max(), 100)
possible_risks = []

for target in possible_returns:
    # Find minimum risk for each return target
    constraints = (
        {'type': 'eq', 'fun': lambda w: np.sum(w) - 1},
        {'type': 'eq', 'fun': lambda w: get_portfolio_return(w) - target}
    )

    result = minimize(get_portfolio_risk,
                     start_guess,
                     method='SLSQP',
                     bounds=weight_limits,
                     constraints=constraints)

    if result.success:
        possible_risks.append(result.fun)
    else:
        possible_risks.append(np.nan)
```

```
plt.figure(figsize=(12, 6))
plt.plot(possible_risks, possible_returns, 'blue', label='Efficient
Frontier')
plt.scatter(safe_risk, safe_return, color='red', s=200, marker='*',
label='Minimum Variance Portfolio')
plt.scatter(best_risk, best_return, color='green', s=200, marker='*',
label='Tangency Portfolio')
plt.xlabel('Yearly Risk')
plt.ylabel('Yearly Return')
plt.title('Investment Portfolio Options')
plt.legend()
plt.grid(True)
plt.show()

# Optional: Show how to distribute money
print("Safest Portfolio Distribution:")
for stock, weight in zip(df.columns, safe_weights.round(4)):
    print(f"{stock}: {weight*100:.1f}%")

print("\nBest Ratio Portfolio Distribution:")
for stock, weight in zip(df.columns, best_weights.round(4)):
    print(f"{stock}: {weight*100:.1f}%")
```