VOLATILITY VOYAGE

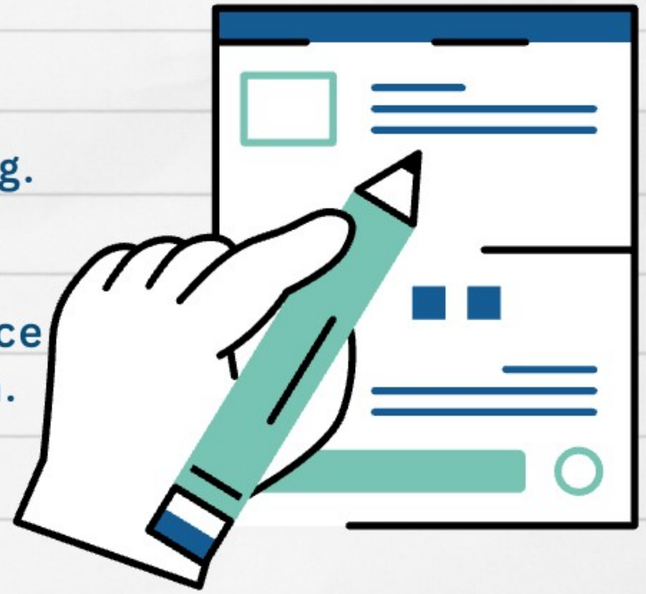
FINANCE AND ANALYTICS CLUB
SUMMER PROJECT 2025

ANURAG || ARIHANT || KETHAN || SHIVAM

INTRODUCTION TO THE PROJECT

Project Introduction: Volatility-Based Trading Strategy

This 8-week project introduces mentees to volatility modeling and its use in building data-driven, risk-aware trading strategies. Till now participants have leveraged Autoregressive models to forecast market volatility and design strategies that adapt position sizing, apply stop-losses, and use dynamic hedging. They will also develop a market regime detection module using indicators like rolling volatility, EWMA to identify bull and bear phases. The project shall culminate in a Python-based backtesting engine to evaluate strategy performance using metrics such as Sharpe Ratio, Value at Risk (VaR), and maximum drawdown.



Deliverables

1. Volatility-Based Trading System: A strategy that uses autoregressive volatility forecasts to guide trade decisions, manage risk, and dynamically adjust position sizes based on expected market conditions.
2. Market Regime Detection Tool: A module to classify market phases using volatility-based indicators.
3. Backtesting and Performance Engine: A Python tool to evaluate the strategy's effectiveness using historical data and financial performance metrics.

PROJECT TIMELINE

Week 1&2

Intoduction to volatility and to backtesting and trade metrics

Week 3

Autoregressive Volatility Modeling

Week 4

Volatility based strategy design

Week 5

Risk Management and dynamic hedging

Week 6

Backtesting Final Strategy and evaluation

Week 7&8

Integration, Documentation and final presentation

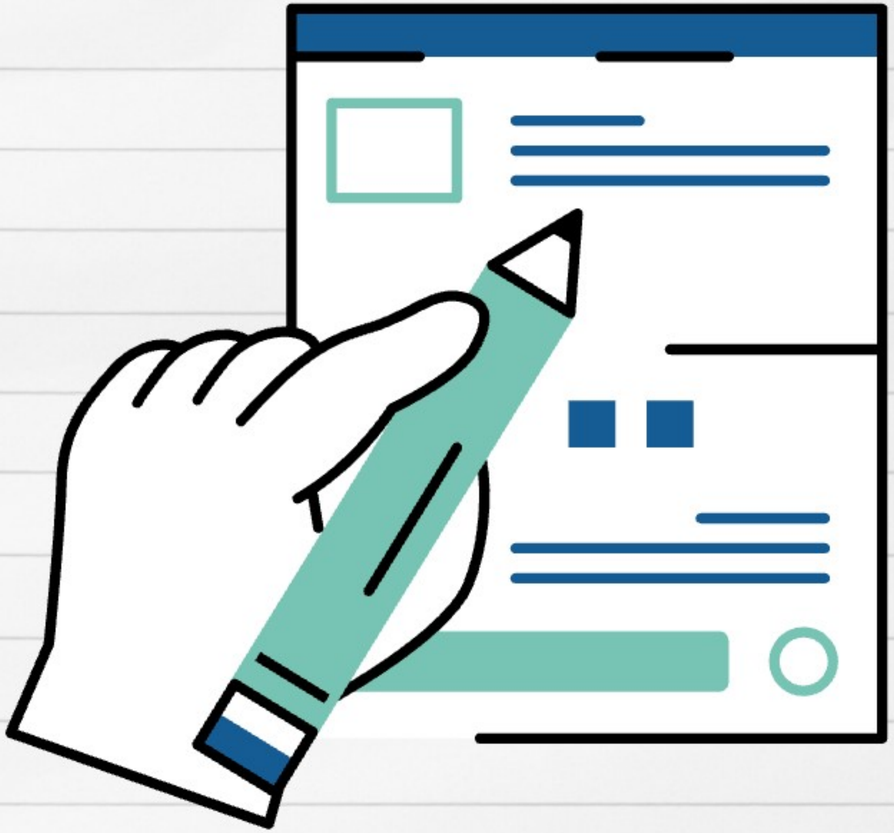
AN OVERVIEW OF WHAT'S BEEN DONE IN THE ASSIGNMENTS TILL NOW

Assignment 1: Trend and Volatility Analysis
https://colab.research.google.com/drive/184k8a4YF3gDR1e3X6WNdk0bNXe6ZKwaS

Assignment 2: Volatility-Integrated Trading Strategies
https://colab.research.google.com/drive/1S__ld_FPQMZ3p351QQU73zQgOySjBqOR

Assignment 3: Volatility Voyage – Risk Management in Trading Strategy -
https://colab.research.google.com/drive/1gNXRsijtETm7dyV0y17Pq52h7CgQcGjR?usp=sharing

Assignment 4: Portfolio Analysis Using Indian Stocks & Volatility Forecasting
https://drive.google.com/drive/folders/1AoMeYjKwMcF8S0YSgFdnm5LjNWJtc2E7



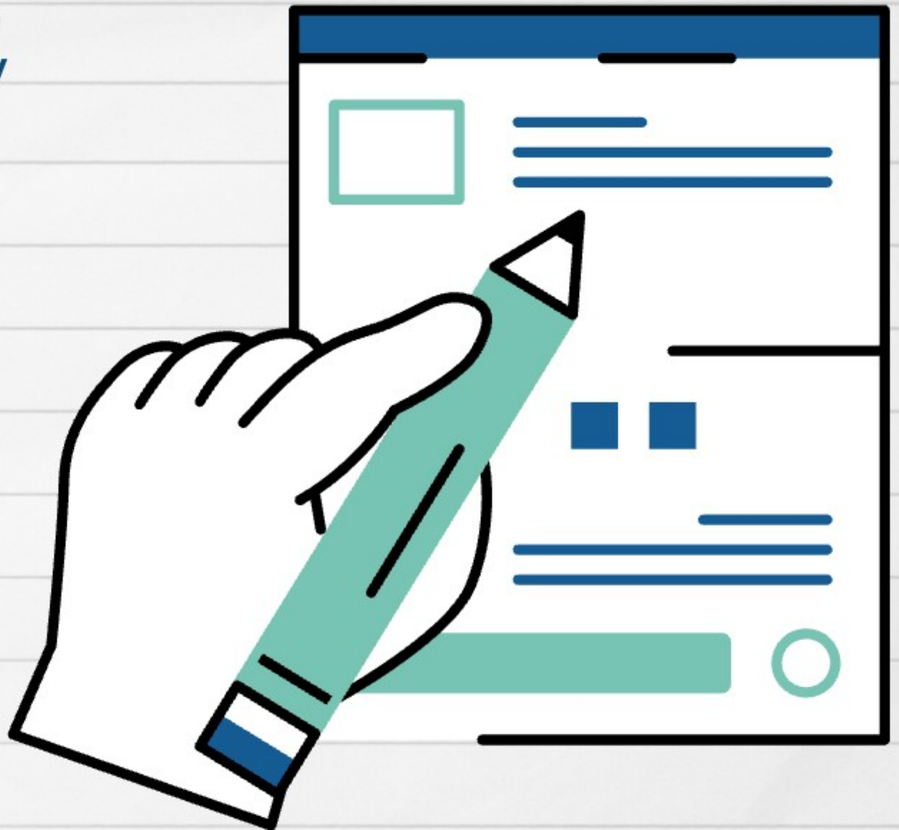
INTRODUCTION TO QUANTITATIVE TRADING STRATEGIES

Decoding Quantitative Trading:

Quantitative trading strategies use mathematical and statistical tools to make data-driven trading decisions. These methods rely on market data — primarily prices — and apply formulas to detect patterns or trends. In this project, we studied:

- Trend-following indicators: SMA and EMA
- Volatility-aware trading techniques
- Price-band-based signal generation

The strategies progress from simple moving averages to more adaptive, volatility-sensitive rules that help traders enter and exit markets systematically.



FOLLOWING THE MARKET'S DIRECTION

Simple Moving Average (SMA):

-> Averages past n days of prices equally

-> Smooths short-term noise

-> Useful to identify the general trend

Exponential Moving Average (EMA):

-> Assigns more weight to recent prices

-> Responds quicker to trend changes

-> Commonly used with two intervals (e.g., EMA-20, EMA-50)

What We Did (Assignment 1):

```
df['SMA'] = df['close'].rolling(window=20).mean()

df['EMA_20'] = df['close'].ewm(span=20, adjust=False).mean()
df['EMA_50'] = df['close'].ewm(span=50, adjust=False).mean()
```

- These strategies only followed price trends and didn't include volatility.

FIRST TRADING STRATEGY–EMA BASED SIGNAL GENERATION

Logic:

- Uses 14-day EMA of closing prices
- Generates Buy/Sell signals based on crossover

Rules:

- Buy Signal: When current price crosses above 14-EMA
- Sell Signal: When current price crosses below 14-EMA and previous price was higher than the EMA .
- Hold: If no crossover occurs

Why It Works:

- Captures momentum by identifying shifts in short-term price direction
- Simple yet effective for trending markets

```python
def ema_crossover_strategy(data):
    # Calculate 14-period Exponential Moving Average (EMA) of the Close price
    data['EMA14'] = data['Close'].ewm(span=14, adjust=False).mean()

    # Initialize signal list: 1 = Buy, -1 = Sell, 0 = Hold
    signals = [0]
    # Loop through each row starting from the second one
    for i in range(1, len(data)):
        # Previous and current Close prices
        prev_close = data['Close'].iloc[i-1]
        curr_close = data['Close'].iloc[i]

        # Previous and current EMA values
        prev_ema = data['EMA14'].iloc[i-1]
        curr_ema = data['EMA14'].iloc[i]

        # Buy Signal: Current price crosses above EMA, Previous was below EMA
        if curr_close > curr_ema and prev_close < prev_ema:
            signals.append(1)
        # Sell Signal: Current price crosses below EMA, Previous was above EMA
        elif curr_close < curr_ema and prev_close > prev_ema:
            signals.append(-1)
        # No signal
        else:
            signals.append(0)

    # Ensure that no Sell signals are present before the first Buy
    buy_found = False
    for i in range(len(signals)):
        if signals[i] == 1:
            buy_found = True  # Set flag after first Buy signal
        if not buy_found and signals[i] == -1:
            signals[i] = 0  # Cancel Sell signals before first Buy

    # Add the generated signals to the original DataFrame
    data['Signal'] = signals

    return data
```

SMARTER STRATEGIES WITH DYNAMIC PRICE BANDS

STRATEGY 2 – EMA + HIGH-LOW BAND:

- **What it does:** Adds a buffer zone (band) around the EMA using the day's price range (High - Low).

- Buy Signal: When the price breaks above EMA + (High - Low)
- Sell Signal: When the price drops below EMA - (High - Low)

- **Why it's better:**
- Filters out minor fluctuations that might falsely trigger the basic EMA strategy.
- Only triggers trades during strong intraday price movements.
- Helps avoid noise and sideways market whipsaws.

```python
def enhanced_ema_band_strategy_with_management(data):
    # Step 1: Calculate 14-period Exponential Moving Average (EMA)
    data['EMA14'] = data['Close'].ewm(span=14, adjust=False).mean()

    # Step 2: Estimate volatility as the difference between High and Low prices
    data['Volatility'] = data['High'] - data['Low']

    # Step 3: Create dynamic Buy and Sell bands around EMA using volatility
    data['Buy_Band'] = data['EMA14'] + data['Volatility']  # Upper band
    data['Sell_Band'] = data['EMA14'] - data['Volatility']  # Lower band

    # Step 4: Initialize signal list - 1 = Buy, -1 = Sell, 0 = No signal
    signals = [0]
    # Step 5: Generate signals based on price crossing the bands
    for i in range(1, len(data)):
        price = data['Close'].iloc[i]
        upper_band = data['Buy_Band'].iloc[i]
        lower_band = data['Sell_Band'].iloc[i]
        prev_price = data['Close'].iloc[i-1]
        prev_upper = data['Buy_Band'].iloc[i-1]
        prev_lower = data['Sell_Band'].iloc[i-1]

        # Buy Signal: Price crosses above upper band (from below)
        if price > upper_band and prev_price <= prev_upper:
            signals.append(1)

        # Sell Signal: Price crosses below lower band (from above)
        elif price < lower_band and prev_price >= prev_lower:
            signals.append(-1)

        # No signal
        else:
            signals.append(0)

    # Step 6: Remove any Sell signals before the first Buy signal
    buy_found = False
    for i in range(len(signals)):
        if signals[i] == 1:
            buy_found = True  # Once a Buy is found, future Sells are allowed
        if not buy_found and signals[i] == -1:
            signals[i] = 0  # Cancel early Sell signals

    # Step 7: Suppress repeated consecutive Buy or Sell signals
    last_signal = 0
    for i in range(len(signals)):
        # If same signal is repeated consecutively, suppress it
        if signals[i] == 1 and last_signal == 1:
            signals[i] = 0
        elif signals[i] == -1 and last_signal == -1:
            signals[i] = 0
        # Update the last non-zero signal
        elif signals[i] != 0:
            last_signal = signals[i]

    # Step 8: Attach the signal column to the original DataFrame
    data['Signal'] = signals

    return data
```

SMARTER STRATEGIES WITH DYNAMIC PRICE BANDS

STRATEGY 3 — EMA WITH ATR-BASED BANDS (ATR BREAKOUT)

- **What it does**: Uses the 14-day Average True Range (ATR) to create a volatility-sensitive band around EMA.
- Buy Signal: Price > EMA + 1.5 × ATR
- Sell Signal: Price < EMA - 1.5 × ATR

- Why it's more efficient:
- ATR accounts for broader volatility than just daily range (considers gaps from previous closes too).
- Dynamically adjusts band width based on recent market conditions — expands in volatile periods, contracts in calm ones.
- Reduces unnecessary trades during low-volatility periods, increases precision during breakouts.

```python
def enhanced_atr_volatility_band_strategy(data, k=1.5):
    # Step 1: Calculate the 14-period Exponential Moving Average (EMA) of the close price
    data['EMA14'] = data['Close'].ewm(span=14, adjust=False).mean()

    # Step 2: Compute True Range (TR) components
    prev_close = data['Close'].shift(1)  # Previous day's close
    tr1 = data['High'] - data['Low']     # High - Low of the current day
    tr2 = abs(data['High'] - prev_close) # High - Previous close
    tr3 = abs(data['Low'] - prev_close)  # Low - Previous close

    # Step 3: Take the maximum of the three TR components to get the True Range
    tr = np.maximum(tr1, np.maximum(tr2, tr3))

    # Step 4: Calculate Average True Range (ATR) using 14-period EMA
    atr = tr.ewm(span=14, adjust=False).mean()

    # Step 5: Define upper and lower bands around EMA using ATR scaled by factor k
    buy_band = data['EMA14'] + k * atr   # Upper band for Buy signal
    sell_band = data['EMA14'] - k * atr  # Lower band for Sell signal

    # Step 6: Initialize signal list — 1 = Buy, -1 = Sell, 0 = Hold
    signals = [0]  # First row can't have a signal due to lack of previous data

    # Step 7: Loop through data to generate crossover-based signals
    for i in range(1, len(data)):
        price = data['Close'].iloc[i]
        prev_price = data['Close'].iloc[i-1]

        # Buy signal: Price crosses above upper ATR band
        if price > buy_band.iloc[i] and prev_price <= buy_band.iloc[i-1]:
            signals.append(1)
        # Sell signal: Price crosses below lower ATR band
        elif price < sell_band.iloc[i] and prev_price >= sell_band.iloc[i-1]:
            signals.append(-1)
        # No crossover, hold
        else:
            signals.append(0)

    # Step 8: Suppress Sell signals before first Buy (to avoid shorting)
    buy_found = False
    for i in range(len(signals)):
        if signals[i] == 1:
            buy_found = True  # Once Buy is found, allow Sell
        if not buy_found and signals[i] == -1:
            signals[i] = 0    # Suppress early Sell signals

    # Step 9: Remove repeated consecutive Buy or Sell signals
    last_signal = 0
    for i in range(len(signals)):
        if signals[i] == 1 and last_signal == 1:
            signals[i] = 0  # Avoid repeated Buy
        elif signals[i] == -1 and last_signal == -1:
            signals[i] = 0  # Avoid repeated Sell
        elif signals[i] != 0:
            last_signal = signals[i]  # Update last non-zero signal

    data['Signal'] = signals
    return data
```

BACKTESTING ENGINE AND EVALUATION METRICS

We can’t really deploy anything before rigorously testing it.

After generating buy-sell signals via various strategies, it is essential that we put the strategies to test by using them on historical OHLCV data.

We also evaluate the strategy via various metrics, giving us an understanding of how profitable it is and its level of risk

BACKTESTING ENGINE

A backtesting engine simulates how a trading strategy would have performed on historical stock market data. It ensures that decisions are made using only past information, helping traders assess strategy effectiveness, risk, and reliability before deploying it in real markets.

Metrics for evaluating a strategy after simulation:

- Benchmark Returns (%)
- Net Profit (%)
- Gross Profit (%)
- Number of Trades (Total, Wins, Losses)
- Maximum and Average Holding Time
- Maximum and Average Drawdown (%)
- Maximum and Average Dip (%)
- Sharpe Ratio (risk-free rate = 0)

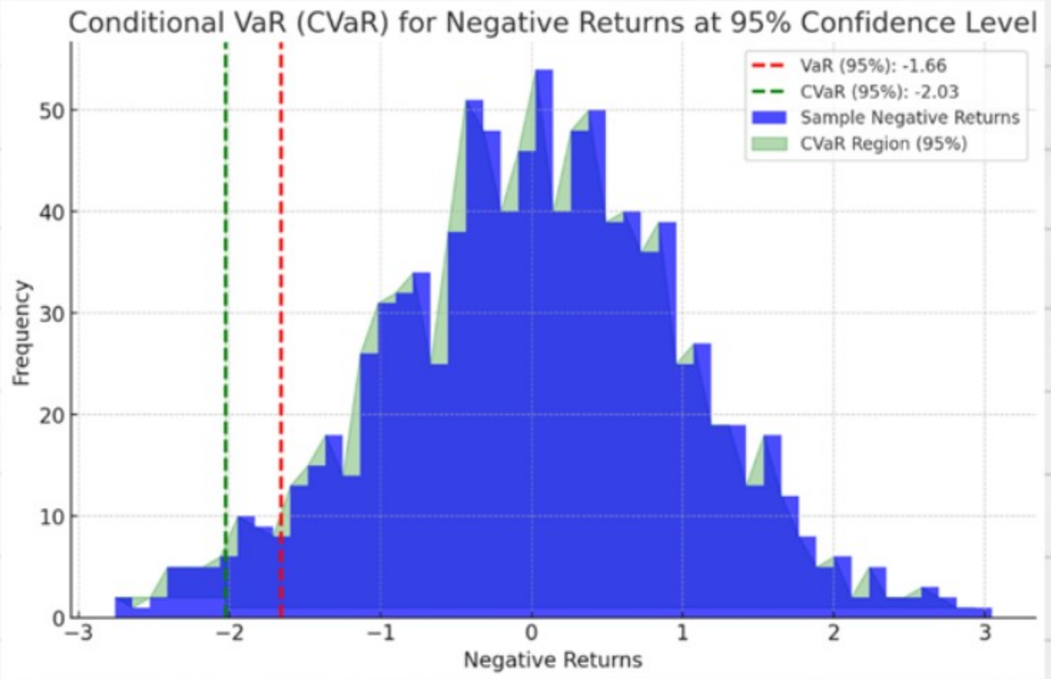
EVALUATION METRICS

- Benchmark Returns (%): The percentage return of a reference index or asset over the same time period for comparison.
- Net Profit (%): The total return after accounting for all losses and costs, expressed as a percentage of initial capital.
- Gross Profit (%): The sum of all positive returns from winning trades before subtracting losses or costs.
- Number of Trades (Total, Wins, Losses): Total trades executed along with the count of profitable and unprofitable ones.
- Maximum and Average Holding Time: The longest and average duration for which trades were held open.
- Maximum and Average Drawdown (%): The worst and average peak-to-trough declines in equity value during the trading period.
- Maximum and Average Dip (%): The largest and average temporary loss in individual trades before closing.
- Sharpe Ratio: The average return per unit of volatility, measuring risk-adjusted performance.

*Two important metrics especially in our case will be discussed in detail after this, they are **VaR** and **CVaR**.*

VAR AND CVAR

- **Value at Risk (VaR)**, estimates the maximum potential loss in a portfolio over a given time period at a certain confidence level (e.g., 95%). It answers the question: “What’s the worst-case loss I can expect, within reason?”
- **Conditional Value at Risk (CVaR)**, also known as Expected Shortfall, goes a step further—it measures the **average loss** in the worst-case scenarios beyond the VaR threshold, giving a more complete picture of extreme risk.



MEASURING VAR AND CVAR

- Value at Risk (VaR) is typically measured using percentiles of historical return data. For example, a 95% VaR is the **5th percentile** of the return distribution — meaning there's a 5% chance the loss will exceed this value.
- Conditional Value at Risk (CVaR) — or Expected Shortfall — is calculated as the **average of all losses that fall below the VaR threshold**. So, if VaR is the 5th percentile, CVaR is the **mean of returns in the worst 5% of outcomes**, providing a clearer view of extreme downside risk

```
# Set confidence level
confidence_level = 0.95
alpha = 1 - confidence_level  # e.g., 0.05

# Calculate historical VaR (at 95% level → 5th percentile)
VaR = np.percentile(returns, 100 * alpha)

# Calculate CVaR: mean of losses worse than the VaR
CVaR = returns[returns <= VaR].mean()
```

A simple code to measure
VaR and CVaR given actual data

RISK MANAGEMENT AND CAPITAL ALLOCATION

"Risk comes from not knowing what you're doing." — Warren Buffett

Every trading strategy comes with their own risks. So it is important for us to manage the risk as per the trader's risk taking capacity.

There are various ways to manage risk, like stop loss or take profit. Or dynamic allocating capital according to the trader's risk taking capacity.

Let's take a look into it.

RISK MANAGEMENT

Investing of any kind generally comes with 2 types of risks:

- **Systematic Risk:** It is a Market-wide risk that affects the entire economy or financial system. It is associated with Inflation, interest rate changes, wars, market crashes, recession, etc. To mitigate it we can use VIX(fear index) or Beta to dynamically allocate capital for trading. Take profit or Stop Loss can also be used.

- **Unsystematic Risk:** It is Company- or industry-specific risk; unique to a particular firm or sector. It is associated with Poor earnings, lawsuits, fraud, sector collapse, etc. To mitigate it, investing in companies with strong fundamentals are required along with portfolio diversification.

DYNAMIC CAPITAL ALLOCATION

Implementation Approach

- Risk-Based Framework: Adjust portfolio weights based on real-time risk metrics (e.g., volatility, VIX levels, drawdown thresholds).
  for example,
  Allocate 100% capital when VIX < Mean + 0.5×SD, Allocate 75% capital when Mean + 0.5×SD ≤ VIX < Mean +1.5×SD and Allocate 50% capital when VIX ≥ Mean +1.5×SD
- Signal-Driven Allocation: Use market signals (momentum, mean reversion, macro indicators) to dynamically shift capital between asset classes.

Benefits

- Enhanced Risk Management: Proactively reduces exposure during high-risk periods, preserving capital.
- Enhanced Return Consistency: Avoids over-leveraging during market stress, leading to smoother equity curves.
- Improved Risk-Adjusted Metrics: Lower drawdowns, better Sharpe Ratio, VaR & CVaR performance.
- Adaptability: Responds to changing market regimes, improving long-term portfolio resilience.

Sample applications of Dynamic Allocation

```python
vix_mean = vix_series.mean()
vix_std = vix_series.std()

threshold_1 = vix_mean + 0.5 * vix_std
threshold_2 = vix_mean + 1.5 * vix_std

# VIX momentum
vix_momentum = vix_series.pct_change().fillna(0)

# Allocation function
def hybrid_allocation(vix, vix_change):
    # Base allocation based on VIX level
    if vix < threshold_1:
        allocation = 1.0
    elif vix < threshold_2:
        allocation = 0.75
    else:
        allocation = 0.5

    # Apply VIX momentum adjustment
    if vix_change < 0:
        # reward: increase allocation by one step (if possible)
        if allocation == 0.5:
            allocation = 0.75
        elif allocation == 0.75:
            allocation = 1.0
    elif vix_change > 0.05:
        # penalize: decrease allocation by one step (if possible)
        if allocation == 1.0:
            allocation = 0.75
        elif allocation == 0.75:
            allocation = 0.5

    return allocation

allocation_series = pd.Series([
    hybrid_allocation(v, vc) for v, vc in zip(vix_series, vix_momentum)],
    index=vix_series.index
)
```

using VIX

```python
window_vol = 20
realized_vol = returns.rolling(window_vol).std()

# Normalize volatility: higher vol -> lower allocation
vol_allocation = 1 / (realized_vol + 1e-6)
vol_allocation = vol_allocation / vol_allocation.max()  # Normalize to [0,1]
vol_allocation = 0.5 + 0.5 * vol_allocation  # Scale to [0.5, 1]

# Computing drawdown
cumulative_returns = (1 + returns).cumprod()
rolling_max = cumulative_returns.cummax()
drawdown = (cumulative_returns - rolling_max) / rolling_max

# Normalize drawdown: deeper drawdown -> lower allocation
drawdown_allocation = 1 + drawdown.clip(-0.5, 0)  # drawdown capped at -50%

# Combining both allocations
allocation_series = np.minimum(vol_allocation, drawdown_allocation)

# Output
allocation_df = pd.DataFrame({
    'Returns': returns,
    'Realized Vol': realized_vol,
    'Vol Allocation': vol_allocation,
    'Drawdown Allocation': drawdown_allocation,
    'Final Allocation': allocation_series
})
```

using combination of
volatility and drawdown

PORTFOLIO ANALYSIS USING INDIAN STOCK MARKET DATA

Objective:- In assignment 4 the objective was to construct and evaluate portfolios using historical monthly data from Indian equity markets.We aim to calculate key performance metrics such as expected returns, volatility, portfolio variance, and the Sharpe ratio. Using these, we will visualise the Efficient Frontier to identify the most optimal portfolios, including the minimum variance and tangency portfolios, and assess how different weight combinations affect risk and return.

Methodology:

1. we have chosen 3 actively traded Indian companies from different sectors to ensure diversification:-
   a) Reliance Industries b) Infosys c) ICICI Bank
2. Period: Jan 2020 – May 2023 (monthly)
3. Calculated the expected return (mean), volatility (standard deviation), and the covariance matrix to evaluate individual and joint risk levels.
4. Then we formulated 5 portfolios by varying weight combinations and calculated portfolio return, standard deviation and sharpe ratio for each of them
5. Then we plotted the Efficient Frontier to visualize the risk-return trade-off of all portfolios.

PORTFOLIO METRICS

```python
import numpy as np

# Risk-free rate (monthly)
rf = 0.005 # used to compute the sharpe ratio.

# Store results
results = []

# Loop through each portfolio
for w in portfolios:
    weights = np.array(w)
    port_return = np.dot(weights, expected_returns)
    port_variance = np.dot(weights.T, np.dot(cov_matrix, weights))
    port_std = np.sqrt(port_variance)
    sharpe_ratio = (port_return - rf) / port_std
    results.append({
        'Weights': weights,
        'Return': port_return,
        'Std Dev': port_std,
        'Sharpe Ratio': sharpe_ratio
    })

port_df = pd.DataFrame(results)
print(port_df)
```

	Weights	Return	Std Dev	Sharpe Ratio
0	[0.33, 0.33, 0.34]	0.019051	0.070427	0.199510
1	[0.5, 0.3, 0.2]	0.019426	0.072833	0.198073
2	[0.2, 0.5, 0.3]	0.018842	0.070990	0.194987
3	[0.1, 0.2, 0.7]	0.018452	0.077983	0.172496
4	[0.4, 0.4, 0.2]	0.019249	0.070679	0.201607

Summary Metrics:

Ticker	Expected Return	Volatility
ICICIBANK.NS	0.020418	0.100425
INFY.NS	0.018649	0.088355
RELIANCE.NS	0.018115	0.091063

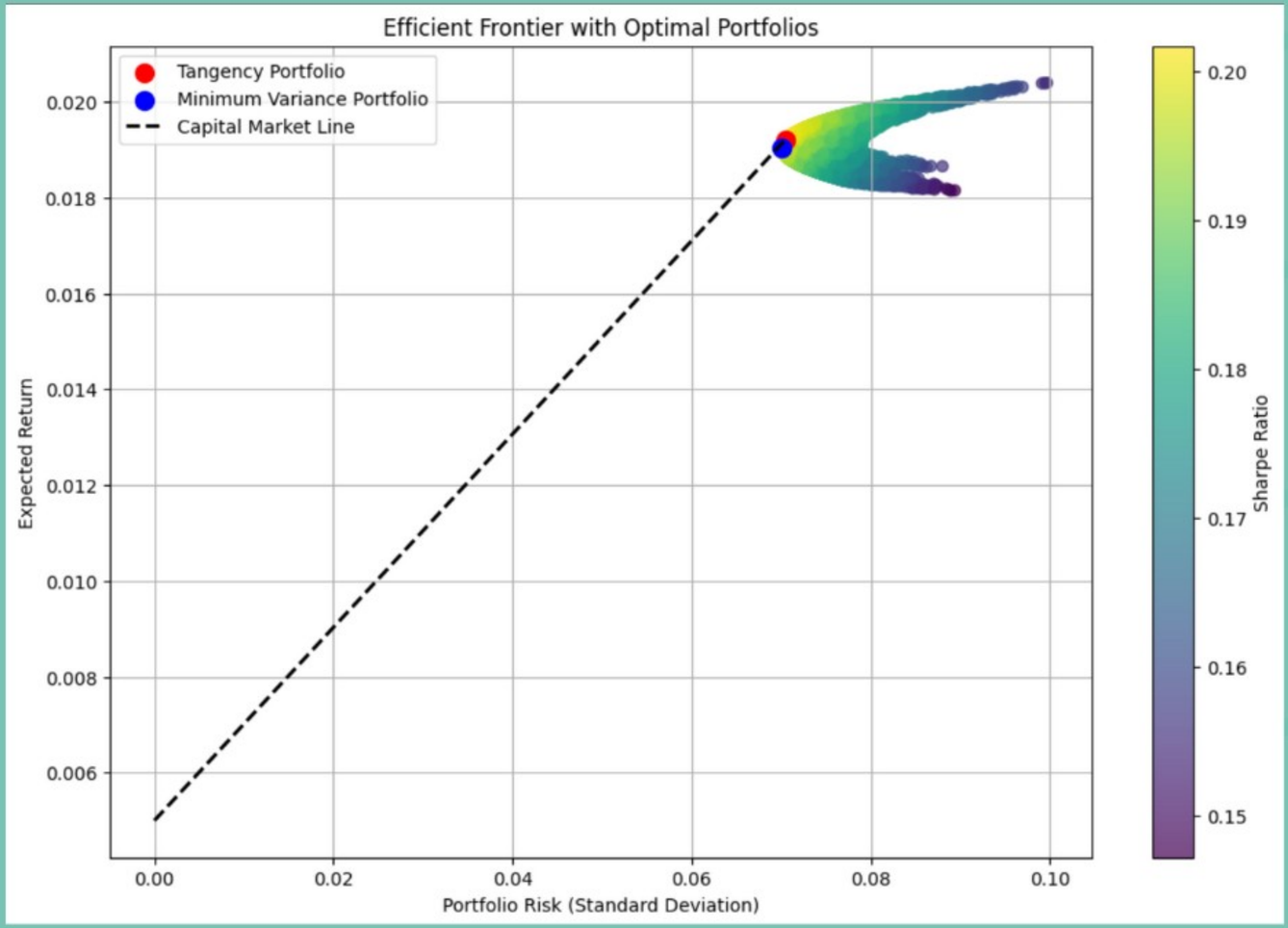
Covariance Matrix:

Ticker	ICICIBANK.NS	INFY.NS	RELIANCE.NS
ICICIBANK.NS	0.010085	0.002048	0.003442
INFY.NS	0.002048	0.007807	0.003719
RELIANCE.NS	0.003442	0.003719	0.008292

- 5 PORTFOLIOS TESTED WITH DIFFERENT WEIGHT COMBINATIONS OF THREE STOCKS.
- METRICS CALCULATED FOR EACH: EXPECTED RETURN, RISK (STD DEV), SHARPE RATIO.
- AMONG THE 5 MANUALLY CREATED PORTFOLIOS, THE ONE WITH WEIGHTS [0.4, 0.4, 0.2] HAD THE HIGHEST SHARPE RATIO (0.2016), MAKING IT THE TANGENCY PORTFOLIO.
- TANGENCY PORTFOLIO OPTIMISED THE RISK-RETURN TRADEOFF BY SLIGHTLY FAVOURING RELIANCE AND INFOSYS (LOWER-RISK STOCKS) WHILE STILL CAPTURING ENOUGH RETURN FROM ICICI BANK, RESULTING IN THE HIGHEST SHARPE RATIO.
- THE PORTFOLIO WITH WEIGHTS [0.33, 0.33, 0.34] HAD THE LOWEST STANDARD DEVIATION OF 0.0704, MAKING IT THE MINIMUM VARIANCE PORTFOLIO AMONG 5 TEST PORTFOLIOS.
- MINIMUM VARIANCE PORTFOLIO ACHIEVED THE LOWEST RISK BY BALANCING ALL THREE STOCKS NEARLY EQUALLY, REDUCING OVERALL VOLATILITY THROUGH DIVERSIFICATION — EVEN THOUGH NONE OF THE INDIVIDUAL STOCKS HAD THE LOWEST VOLATILITY.
- “ICICI BANK HAD THE HIGHEST RETURN AND VOLATILITY, BUT OVERALL PORTFOLIO RETURN DID NOT ALWAYS INCREASE WITH VOLATILITY — HIGHLIGHTING THE ROLE OF DIVERSIFICATION.“

EFFICIENT FRONTIER

EFFICIENT FRONTIER PLOT




```
# Generate many random portfolios using these three stocks
num_portfolios = 10000
all_weights = [] # portfolio weights for the 3 stocks
port_returns = [] # expected return of each portfolio.
port_risks = [] # standard deviation (risk) of each portfolio
sharpe_ratios = [] # sharpe ratio of each portfolio.
```

```
# Identify max Sharpe Ratio (Tangency Portfolio)
max_sharpe_idx = sharpe_ratios.argmax() # gives the index of the highest Sharpe ratio
tangent_ret = port_returns[max_sharpe_idx] # stores return of this
tangent_risk = port_risks[max_sharpe_idx] # stores standard deviation

# Identify Minimum Variance Portfolio (lowest standard deviation)
min_var_idx = port_risks.argmin() # gives the index of the lowest risk
min_var_ret = port_returns[min_var_idx]
min_var_risk = port_risks[min_var_idx]

# Plot: Creates the scatter plot of all 10,000 portfolios
plt.figure(figsize=(12, 8))
plt.scatter(port_risks, port_returns, c=sharpe_ratios, cmap='viridis', alpha=0.7)
plt.colorbar(label='Sharpe Ratio')
plt.xlabel('Portfolio Risk (Standard Deviation)')
plt.ylabel('Expected Return')
plt.title('Efficient Frontier with Optimal Portfolios')

# Plot Tangency Portfolio
plt.scatter(tangent_risk, tangent_ret, c='red', s=100, label='Tangency Portfolio')
#Plots the best risk-adjusted portfolio in red with a large marke

# Plot Minimum Variance Portfolio
plt.scatter(min_var_risk, min_var_ret, c='blue', s=100, label='Minimum Variance Portfolio')

# Capital Market Line
cml_x = [0, tangent_risk]
cml_y = [rf, tangent_ret]
plt.plot(cml_x, cml_y, color='black', linestyle='--', linewidth=2, label='Capital Market Line')

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

INSIGHTS & RECOMMENDATION

Diversification Benefits

- Combining stocks from different sectors reduced overall portfolio risk.
- Portfolio volatility was lower than the average volatility of individual stocks.

Impact of Weight Changes

- Increasing weight the highest-return stock generally raised portfolio risk, but not always return — showing that weight decisions must consider diversification."
- Some combinations yielded similar returns but poor Sharpe Ratios due to high volatility

Sharpe Ratio as Decision Tool

- The Sharpe Ratio captured the best return per unit of risk.
- It helped identify the Tangency Portfolio, which lies on the Efficient Frontier and the Capital Market Line.

Recommended Portfolio: Tangency Portfolio

- Highest Sharpe Ratio
- Balanced exposure to energy, IT, and banking sectors
- Optimises return while keeping risk reasonable

Conclusion:-

Based on risk-return optimization, the Tangency Portfolio is the most efficient choice, offering the best balance between performance and volatility. It is recommended for investors seeking growth with managed risk.While ICICI Bank showed the highest standalone return and volatility, the relationship between risk and return across portfolios was not strictly linear. Due to the impact of diversification and inter-stock correlation, some portfolios with moderate weights delivered better Sharpe Ratios despite slightly lower returns. This highlights that portfolio risk is not only a function of individual volatilities, but also of how assets interact."

VOLATILITY FORECASTING:

PROCESS OF PREDICTING THE FUTURE VARIABILITY (STANDARD DEVIATION) OF ASSET RETURNS OVER A SPECIFIED TIME HORIZON

MOTIVATION:

ALIGN PORTFOLIOS TO INVESTOR'S RISK PROFILES

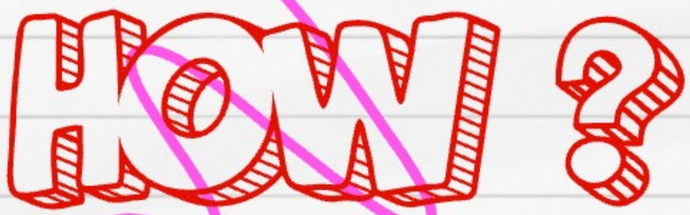
TO MAKE INFORMED TRADING & HEDGING DECISIONS

FOR INVESTORS VOLATILITY IS A DIRECT MEASURE OF RISK

METHODOLOGY:

WE TOOK ALL STOCKS UNDER NIFTY 50 AND CALCULATED THE VOLITILITY OF THEM FROM 1ST JAN 2022 TO 1ST SEPTEMBER 2024 .

USED THE ABOVE DATA TO FORECAST VOLATILITY OF ALL THE 50 STOCKS



WE USED THREE STATIC METHODS TO FORECAST VOLATILITY:

1. ROLLING MEAN FORECAST
2. EXPONENTIALLY WEIGHTED MOVING AVERAGE (EWMA)
3. AUTOREGRESSIVE MODEL OF ORDER 1 (AR(1)) ON VOLATILITY

ROLLING MEAN FORECAST

IT USES THE AVERAGE OF THE PAST N PERIODS' VOLATILITIES TO FORECAST THE NEXT PERIOD'S VOLATILITY

$$\sigma_{\text{forecast}} = \frac{1}{N} \sum_{i=1}^{N} \sigma_{t-i}$$

```
data['vol_forecast_rm'] = data['rolling_vol'].rolling(window=5).mean()
```

ASSUMPTION:

FUTURE VOLATILITY WILL BE SIMILAR TO THE RECENT AVERAGE

PROS:

STABLE AND EASY TO COMPUTE

CONS:

PERFORMS POOR IN VOLATILE PERIODS

INHERENT LAG

EXPONENTIALLY WEIGHTED MOVING AVERAGE (EWMA)

ASSIGNS HIGHER WEIGHT TO RECENT VOLATILITY

HOW ? → A SMOOTHING PARAMETER (LAMBDA) CONTROLS HOW MUCH WEIGHT IS GIVEN TO RECENT DATA.

RESPOND MORE QUICKLY TO CHANGES IN VOLATILITY

$$\text{EWMA}_t = \sqrt{\text{EMA}(r_t^2)} = \sqrt{(1-\lambda)\cdot r_t^2 + \lambda \cdot \text{EWMA}_{t-1}^2}$$

```
data['vol_squared'] = data['returns']**2
data['ewma vol'] = data['vol squared'].ewm(alpha=1 - lambda ).mean() ** 0.5
```

EWMA PROS:

- RESPONSIVE TO RECENT CHANGES
- EFFECTIVE NOISE REDUCTION
- MORE EFFICIENT THAN RMF

EWMA CONS:

- SUSCEPTIBLE TO FALSE SIGNALS
- PARAMETER SELECTION IS CRUCIAL
- EQUAL WEIGHTING OF PAST DATA'S INFLUENCE

AR(1) MODEL ON VOLATILITY

TIME SERIES MODEL THAT FORECASTS FUTURE VALUES BASED ON THE IMMEDIATELY PRECEDING VALUE

EFFICIENT WHEN VOLATILITY IS AUTOCORRELATED(CLUSTERING)

$$\sigma_t = \alpha + \beta\sigma_{t-1} + \varepsilon_t$$

$\sigma(t)$ = FORECASTED VOLATILITY AT TIME T

α = LONG-TERM AVERAGE VOLATILITY

β = AUTOREGRESSIVE COEFFICIENT THAT MEASURES HOW STRONGLY THE PREVIOUS PERIOD'S VOLATILITY ($\Sigma(T-1)$) INFLUENCES THE CURRENT PERIOD'S VOLATILITY.

ε = RANDOM SHOCK OR ERROR TERM FOR THE CURRENT TIME PERIOD

```python
from statsmodels.tsa.ar_model import AutoReg


model = AutoReg(data['rolling_vol'].dropna(), lags=1)
fit = model.fit()
forecast = fit.predict(start=len(data), end=len(data))
```

PROS:

- CAPTURES VOLATILITY CLUSTERING
- SIMPLICITY AND INTERPRETABILITY

CONS:

- FAILS TO CAPTURE THE LEVERAGE EFFECT
- ASSUMES A LINEAR RELATIONSHIP

RESULTS

IN THE 4TH ASSIGNMENT, WE WERE ASKED TO FORECAST VOLATILITY AND THEN SELECT THE TOP 3 STOCKS FROM THE NIFTY 50 WHICH HAD THE LEAST DISTANCE FROM INVESTOR'S RISK AVERSION VALUE AND ASSIGN WEIGHTS TO THEM IN PROPORTION TO THE INVERSE OF THEIR FORECASTED VOLATILITY

	forecasted_vol	normalized_vol	distance	weight	investment
HDFCLIFE.NS	0.017667	0.490273	0.009727	0.340010	340009.68
TECHM.NS	0.018144	0.513945	0.013945	0.331071	331070.72
ADANIPORTS.NS	0.018263	0.519834	0.019834	0.328920	328919.60

INVESTOR'S RISK AVERSION=0.5

WE THEN USED ATR BASED STRATEGY ON THE PORTFOLIO AND BACKTESTED IT, TO OBTAIN THE FOLLOWING BACKTESTING METRICS

```
Benchmark Return: 24.402571818840002 %
Gross Profit: 39.09483602219997 %
Max Holding Time: 103
Average Holding Time: 39.35925925925926
Total Trades: 30
Winning Trades: 13
Losing Trades: 17
Max Drawdown: -18.053750334159737 %
Sharpe Ratio: 0.36303468725542126
```

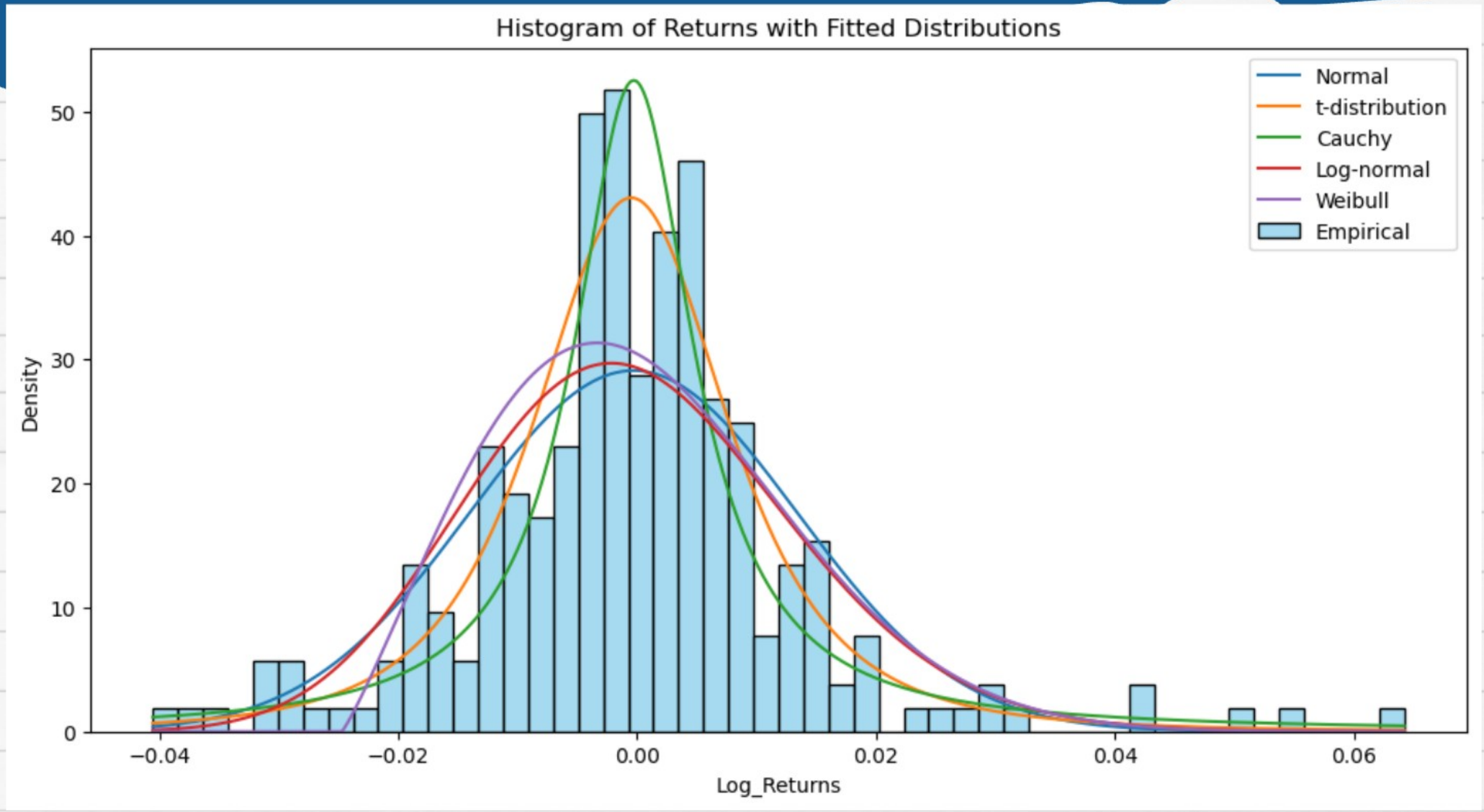
MAXIMUM LIKELIHOOD ESTIMATION –MLE

TO MODEL THE PROBABILITY DISTRIBUTION OF DAILY LOG RETURNS FOR TCS.NS THE FOLLOWING DISTRIBUTIONS WERE SELECTED.

1. NORMAL – SYMMETRIC, LIGHT TAILS.
2. STUDENT'S T – SYMMETRIC, HEAVY TAILS.
3. LOG-NORMAL – RIGHT-SKEWED, HEAVY TAILS.
4. WEIBULL – FLEXIBLE SHAPE FOR SKEW/TAIL VARIATION.
5. CAUCHY – EXTREME FAT TAILS FOR STRESS-TESTING.

```
Normal Distribution Parameters:
  Mean (μ): -0.000316
  Standard Deviation (σ): 0.013706

t-distribution Distribution Parameters:
  Degrees of Freedom (df): 2.7897
  Location (μ): -0.000499
  Scale (σ): 0.008489

Cauchy Distribution Parameters:
  Parameters: (-0.00028555419049953654, 0.006062199471243544)

Log-normal Distribution Parameters:
  Shape (σ): 0.091031
  Location (μ): -0.149088
  Scale (exp(μ)): 0.148155

Weibull Distribution Parameters:
  Shape (c): 2.141878
  Location: -0.024730
  Scale (λ): 0.028675
```

HISTOGRAM OF RETURNS WITH FITTED DISTRIBUTION PDFS



HYPOTHESIS TESTING OF FITTED DISTRIBUTIONS

TO ASSESS THE GOODNESS-OF-FIT FOR EACH MLE-FITTED DISTRIBUTION ON TCS.NS DAILY LOG RETURNS.

METHOD USED:

- KOLMOGOROV–SMIRNOV (KS) TEST IMPLEMENTED VIA STATS.KSTEST().

HYPOTHESES (FOR EACH DISTRIBUTION):

- H0: RETURNS FOLLOW THE FITTED DISTRIBUTION.
- HA: RETURNS DO NOT FOLLOW THE FITTED DISTRIBUTION.

```
Kolmogorov–Smirnov Test: Distribution Ranking by p-value
------------------------------------------------------------
Rank  Distribution    KS Statistic    p-value
------------------------------------------------------------
1     t-distribution  0.0315          0.9591
2     Cauchy          0.0501          0.5439
3     Weibull         0.0915          0.0289
4     Log-normal      0.0922          0.0271
5     Normal          0.0974          0.0165
```

AUTO REGRESSIVE MOVING AVERAGE (ARMA)

The ARMA model is a combination of two models:

- Autoregressive (AR) model : Models the current value using a number of lagged observations (previous time points)
- Moving Average (MA) model : Models the current value as a linear combination of past q error terms.

An ARMA(p, q) process is defined as:

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$$

where ε_t is white noise.

X_t	Value of the time series at time t (the variable being modeled)
ϕ_i (for $i = 1..p$)	AR coefficients representing influence of past values X_{t-i} on X_t
$\sum_{i=1}^{p} \phi_i X_{t-i}$	Autoregressive component: weighted sum of past p values
θ_j (for $j = 1..q$)	MA coefficients representing influence of past noise terms ε_{t-j}
$\sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$	Moving average component: weighted sum of past q noise terms

AUTO REGRESSIVE INTEGRATED MOVING AVERAGE (ARIMA)

The ARIMA model is the extension of the ARMA model by incorporating the idea of differencing to handle non-stationary time-series data.

The ARIMA(p, d, q) process can be expressed as:

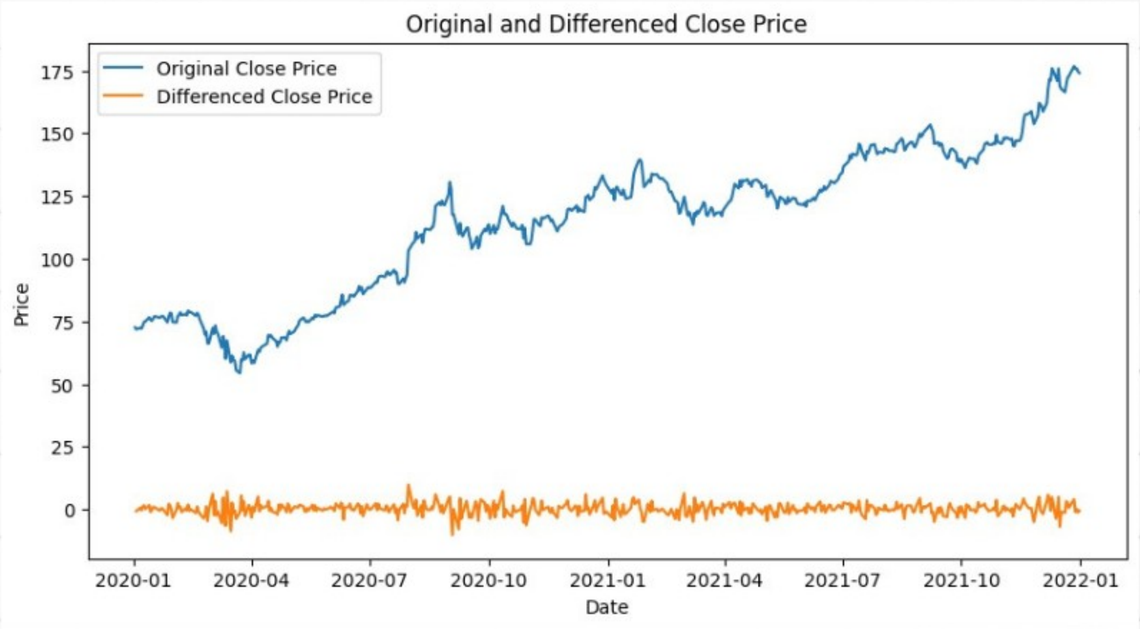
$$\Delta^d X_t = c + \sum_{i=1}^{p} \phi_i \Delta^d X_{t-i} + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$$

where Δ^d denotes differencing d times.

$\Delta^d X_t$	The differenced series at time t, after applying differencing operator d times
$\sum_{i=1}^{p} \phi_i \Delta^d X_{t-i}$	Autoregressive component on differenced data
$\sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$	Moving average component based on past noise terms

- suitable for non-stationary time-series data which can be converted to stationary data after differencing a few times

IMPLEMENTATION




```
total_abs_err = []
actual = (df_future_diff['Close']).head(100).reset_index(drop=True)

for i in range(10):
    model = ARIMA(df_diff['Close'], order=(1, 0, i))
    model_fit = model.fit()
    predictions = model_fit.forecast(steps=100)
    predictions = predictions[:100]

    sum1 = 0
    for j in range(100):
        # Extract the numeric value before adding
        error_val = (actual.iloc[j] - predictions.iloc[j]).abs()
        sum1 += float(error_val)

    print(f"{sum1}  {i}")
    total_abs_err.append(sum1)
```

CONVERTING NON STATIONARY TO STATIONARY TIME SERIES

```
df_diff = df.diff().dropna()
result = adfuller(df_diff['Close'])
print('p-value:', result[1])
```

```
p-value: 0.0
```

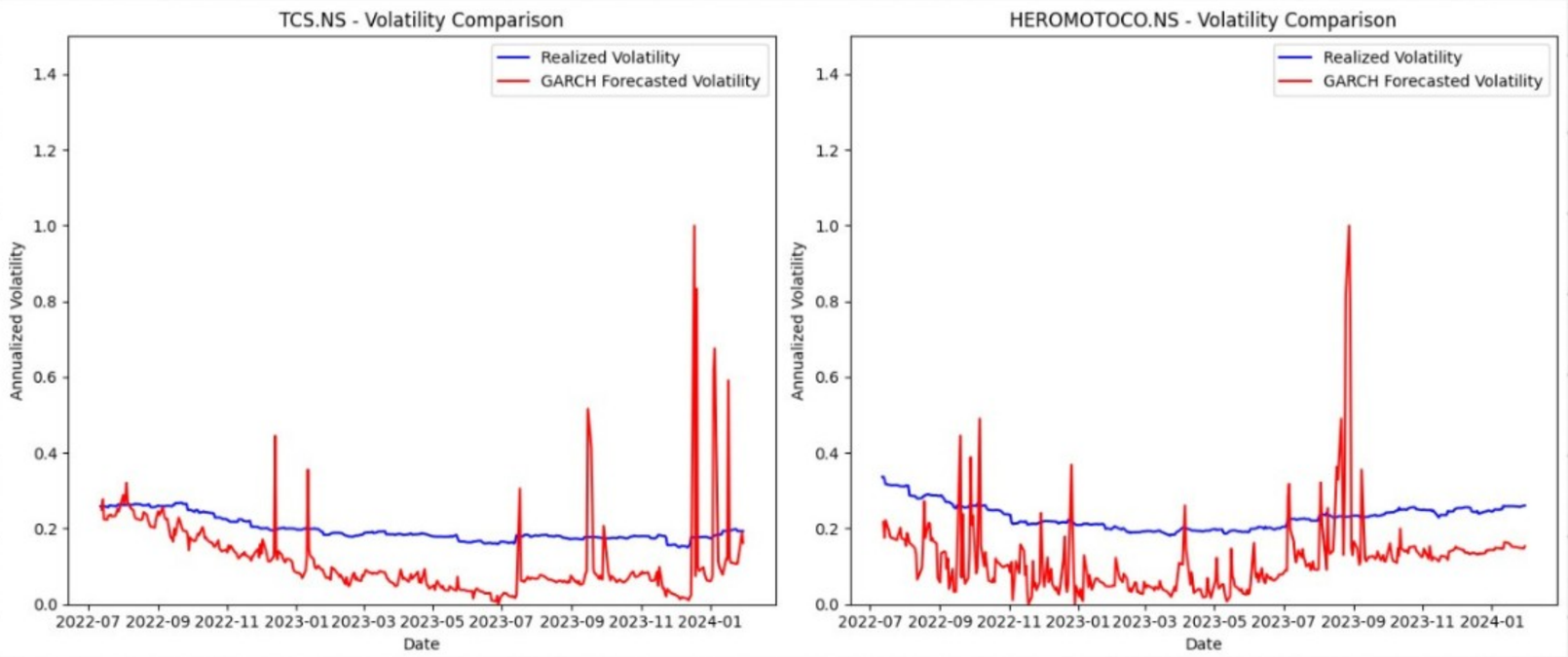
CODE SNIPPET

GENERALIZED AUTOREGRESSIVE CONDITIONAL HETEROSKEDASTICITY(GARCH)

GARCH MODELS TIME-VARYING VOLATILITY IN FINANCIAL RETURNS BY ALLOWING THE CONDITIONAL VARIANCE TO DEPEND ON PAST SQUARED RESIDUALS AND PAST VARIANCES

$$\sigma_t^2 = \omega + \alpha\epsilon_{t-1}^2 + \beta\sigma_{t-1}^2$$

SIGMAT= CONDITIONAL VARIANCE
W= LONG TERM VARIANCE AVERAGE
ALPHA= REACTION PARAMETER
EPSILON(T-1)= THE RESIDUAL SHOCK FROM T-1
BETA= PERSISTENCE PARAMETER



EXPONENTIAL GENERALIZED AUTOREGRESSIVE CONDITIONAL HETEROSKEDASTICITY (EGARCH)

EGARCH extends the standard GARCH model by allowing for asymmetric effects of positive and negative shocks on volatility

$$\ln(\sigma_t^2) = \omega + \alpha(|z_{t-1}| - \mathbb{E}[|z_{t-1}|]) + \gamma z_{t-1} + \beta \ln(\sigma_{t-1}^2)$$

- SIGMA_T^2→ CONDITIONAL VARIANCE AT TIME T
- OMEGA→ BASELINE VOLATILITY LEVEL
- ALPHA→ IMPACT OF SHOCK MAGNITUDE ON VOLATILITY
- GAMMA→ ASYMMETRIC EFFECT OF SHOCK SIGN (LEVERAGE EFFECT)
- BETA → VOLATILITY PERSISTENCE OVER TIME
- Z_T-1→ STANDARDIZED RESIDUAL (SHOCK) FROM PREVIOUS PERIOD

```python
def forecasted(returns):
    window_size = 90
    forecast_horizon = 1

    rolling_forecast = []

    for i in range(window_size, len(returns)):
        train_window = returns[(i - window_size):i]

        model = arch_model(train_window, vol='EGARCH', p=1, o=1, q=1, dist='t')
        res = model.fit(disp='off', options={'maxiter': 100})


        forecast = res.forecast(horizon=forecast_horizon)
        forecasted_var = forecast.variance.values[-1, 0]
        forecasted_vol = np.sqrt(forecasted_var)

        rolling_forecast.append(forecasted_vol)
    mean = np.mean(rolling_forecast)
    return mean,res
```

IMPLEMENTATION

HEDGING

- A RISK MANAGEMENT STRATEGY TO OFFSET POTENTIAL LOSSES BY TAKING AN OPPOSITE POSITION IN A CORRELATED ASSET.

TYPES OF HEDGING

- DERIVATIVES-BASED: OPTIONS, FUTURES, SWAPS
- ASSET-BASED: OPPOSITE POSITIONS IN CORRELATED ASSETS
- FACTOR HEDGING: NEUTRALIZING EXPOSURE TO MARKET FACTORS
- VOLATILITY/CURRENCY/COMMODITY HEDGING

KEY CONCEPTS

BETA — MARKET RISK

$$\beta = \frac{\mathrm{Cov}(R_a, R_m)}{\sigma_m^2}$$

- MEASURES HOW MUCH THE ASSET MOVES WITH THE MARKET.
- HIGH B = MORE VOLATILE THAN MARKET

ALPHA — STRATEGY SKILL

$$\alpha = R_a - [R_f + \beta \cdot (R_m - R_f)]$$

- EXCESS RETURNS BEYOND MARKET RISK.
- POSITIVE A = YOUR STRATEGY BEATS THE MARKET

CORRELATION (P)

$$\rho_{X,Y} = \frac{\mathrm{Cov}(R_X, R_Y)}{\sigma_X \sigma_Y}$$

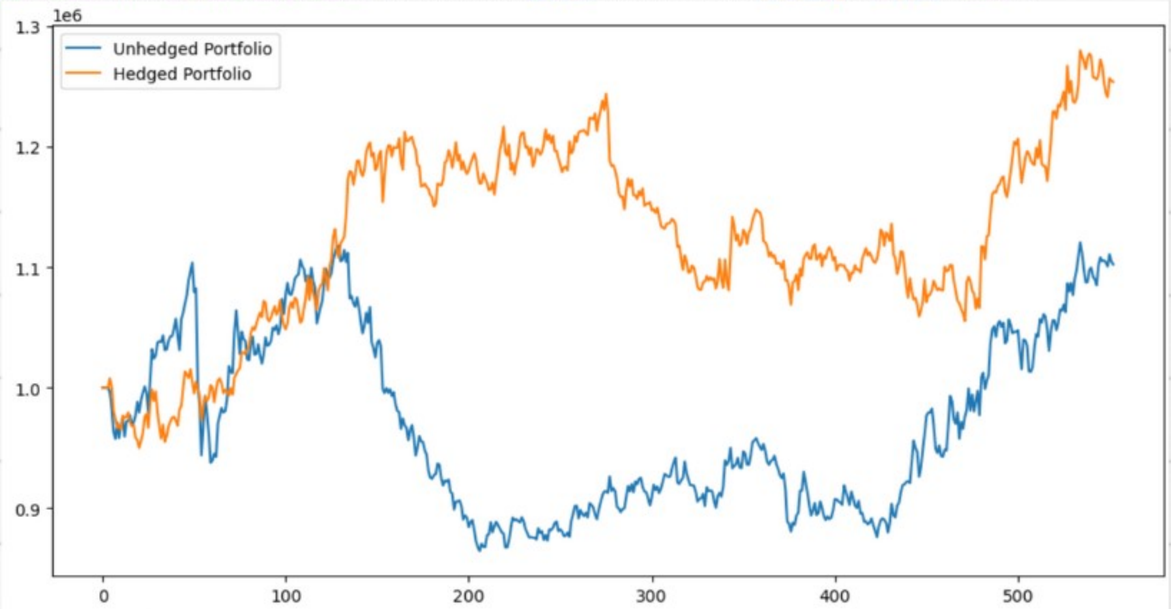
- MEASURES HOW SIMILAR TWO ASSETS MOVE.
- P > 0.7 = STRONG CANDIDATE FOR HEDGE PAIR

HEDGING

This hedging strategy takes a long position in a selected stock and shorts another NIFTY 50 stock that has the highest positive correlation with it. By keeping the investment capital-neutral, it reduces market (beta) risk and focuses on capturing model-based (alpha) returns.

```python
def backtest(stock,df,stock_signals):
  close= list(df[stock])
  opening_capital = 500000
  capital = opening_capital
  portfolio_value = [opening_capital]
  entry_price = 0
  entry_price_hedge = 0
  vol = 0
  no_of_buy_signals = 0
  no_of_sell_signals = 0
  for i in stock_signals:
    if i == 1:
      no_of_buy_signals+=1
    if i == -1:
      no_of_sell_signals+=1

  long_trade = 0
  short_trade = 0
  for i in range(1,len(stock_signals)):
    if stock_signals[i] == 0:
      if long_trade:
        portfolio_value.append(capital+ vol*(close[i] - entry_price))
      elif short_trade:
        portfolio_value.append(capital- vol*(close[i] - entry_price) )
      else:
        portfolio_value.append(capital)
    if stock_signals[i] == -1:
      if long_trade:
        long_trade = 0
        capital+=(vol*(close[i] - entry_price))
        entry_price = 0
        entry_price_hedge = 0
        vol = 0
        hedge_vol = 0
      if no_of_buy_signals > 0:
        entry_price = close[i]
        vol = int(capital/close[i])
        short_trade = 1
      no_of_sell_signals-=1
      portfolio_value.append(capital)
    if stock_signals[i] == 1:
      if short_trade:
        short_trade = 0
        capital-=(vol*(close[i] - entry_price))
        entry_price = 0
        entry_price_hedge = 0
        vol = 0
        hedge_vol = 0
      if no_of_sell_signals > 0:
        entry_price = close[i]
        vol = int(capital/close[i])
        long_trade = 1
      no_of_buy_signals-=1
      portfolio_value.append(capital)
  return portfolio_value
```

```python
stock1_portfolio = backtest(stock1,df,stock1_signals)
hedged_stock1_portfolio = hedged_backtest(stock1,hedge_stock1,df,stock1_signals)
stock2_portfolio = backtest(stock2,df,stock2_signals)
hedged_stock2_portfolio = hedged_backtest(stock2,hedge_stock2,df,stock2_signals)
final_portfolio = []
final_hedged_portfolio = []
for i in range(len(stock1_portfolio)):
  final_portfolio.append(stock1_portfolio[i] + stock2_portfolio[i])
  final_hedged_portfolio.append(hedged_stock1_portfolio[i] + hedged_stock2_portfolio[i])

plt.figure(figsize=(12,6))
plt.plot(final_portfolio,label='Unhedged Portfolio')
plt.plot(final_hedged_portfolio,label='Hedged Portfolio')
plt.legend()
plt.show()

print('******BACKTEST METRICS******')
print(f'Hedged Portfolio Returns: {(final_hedged_portfolio[-1] - final_hedged_portfolio[0])*100/final_hedged_portfolio[0]}%')
print(f'Unhedged Portfolio Returns: {(final_portfolio[-1] - final_portfolio[0])*100/final_portfolio[0]}%')
print(f'Hedged Portfolio Sharpe Ratio: {sharpe_ratio(pd.Series(final_hedged_portfolio).pct_change().dropna(),255,0)}')
print(f'Unhedged Portfolio Sharpe Ratio: {sharpe_ratio(pd.Series(final_portfolio).pct_change().dropna(),255,0)}')
```




```
******BACKTEST METRICS******
Hedged Portfolio Returns: 25.38847281951904%
Unhedged Portfolio Returns: 10.234551934814453%
Hedged Portfolio Sharpe Ratio: 0.80439357825138
Unhedged Portfolio Sharpe Ratio: 0.35479127206215094
```

THANK YOU VERY MUCH!