## **Group Assignment 2: Development Trading Execution Module**

The overall framework will consist of three main components:

1. **Data Pre-processing**: This module will be responsible for preparing and cleaning the data, ensuring it is in the proper format for trading simulations.
2. **Trading Execution**: A class that will instantiate and execute trading strategies based on the pre-processed data. This component should be able to handle different types of trading strategies and ensure accurate execution of trades.
3. **Post-Trade Analysis**: After trading simulation, this module will analyse the returns and performance of the strategy. It will calculate various metrics such as the Sharpe ratio, and generate visualisations to help in assessing the strategy’s effectiveness.

### **Objective**

The objective of this assignment is to develop the "Trading Execution" class that can execute trading strategies based on preprocessed data and signals generated by a strategy builder function. Additionally, create a web interface that allows users to script and run trading strategies directly through a browser.

### **Part 1: Trading Execution Module**

1. **Strategy Builder Function**: Implement a function named strategy\_build that identifies buy and sell signals using a simple moving average (SMA) crossover strategy (9-day SMA crossing over 20-day SMA). This function should update the DataFrame with a "signal" column indicating -1 for sell, 1 for buy, and 0 for hold.
2. **Execution Function**: Develop a run function within the "Trading Execution" class that takes the cleaned DataFrame with SMA signals, along with the ticker name, start date, and end date as inputs. The function should:
   * Loop over the df index which are as date time to -
   * Execute trades based on the SMA signals 1 buy, -1 sell, 0 do nothing (use nested if else conditions, hint : use position variable to store current position and update it in the nested conditions if it is met, store the trade\_open/close price, if it is sell signal and we have bought - calculate returns and store it in df[returns].iloc[i])
   * Implement a stop loss level of 5%
   * Track and store returns in a time-indexed Pandas series
   * Return the returns series to be used for post-trade analysis

This code should be encapsulated within a trading.py file.

### 

### **Part 2: Web Interface Development**

1. **Strategy Scripting Subpage**: Design a subpage under the endpoint Backtester/ that serves as an online Python script editor. This page should allow users to:
   * Write and edit trading strategies in a text editor.
   * Submit the script for execution, which should instantiate the "Trading Execution" class and run the strategy.
2. **Main Webpage Workflow**:
   * **Data Input**: Ask the user for the ticker name, start date, and end date.
   * **Preprocessing Option**: After input, provide an option to preprocess data. Once preprocessing is complete, offer a button to view preprocessing statistics on a new subpage Backtester/preprocess.
   * **Strategy Writing and Execution**: Post-preprocessing, provide an editor for users to write or paste their strategy (python code) and a run button to execute it.
   * **Result Display**: After running the strategy, display trade metrics and performance graphs.

### **Final Submission Requirements**

* **Code**: Submit well-documented main.py and trading.py files that utilize modular code.
* **Web Interface**: Include all files necessary to run the web interface, ensuring it is user-friendly and functional.
* **Demonstration Video**: Produce a video (less than 5 minutes) showing the functionality of your code and web interface, focusing on how the trading strategies can be scripted and executed through the web interface.