Python Dev Challenge

Backtesting Task

Objective:

This task will focus on handling large historical datasets with multiple option expiries across various derivatives (e.g., Nifty, BankNifty, FinNifty, BankEx). You are expected to develop a robust strategy to efficiently manage this data, implement strategies or signal generation (including buy/sell signals with stop-loss), and run a backtest using an open-source backtesting framework.

Task Outline:

1. Step 1: Handling Large Historical Data with Multiple Expiries

The first part of the task is to **design a plan for managing large historical datasets** that contain multiple option expiries for different derivatives such as Nifty, BankNifty, FinNifty, and BankEx. You need to consider the following challenges:

- **Sample Data:** The script should fetch data either from the provided Telegram channel https://2ly.link/1ztTU make sure you are able to extract .feather files and nested zip files or use an alternative open-source source like ICICI Breeze API for historical options/futures data.
- **Large Data Volume:** Data can span several years with multiple expiries, and each expiry may have different strike prices for options.
- **Efficient Data Handling:** You need to design a solution that efficiently loads and processes data to avoid performance bottlenecks.
- **Option Expiry Management:** For options trading strategies, the data must be filtered based on the expiry date and associated with the correct derivative.
- **Multiple Derivatives:** Handle data for different indices and contracts simultaneously.

Your task:

- Come up with a strategy to handle this data efficiently (using techniques like **data partitioning**, **chunk loading**, and **expiry-based filtering**).
- Describe how you would store and retrieve this data (e.g., using **relational databases**, **data lakes**, **Parquet/Feather** formats, or **cloud storage**).
- Explain how you will process the data dynamically based on **different expiries and strategies** (e.g., selecting the correct expiry contracts for BankNifty, Nifty, etc.).

Things to consider:

- How will you structure the data for different indices and derivatives?
- How will you handle rolling options contracts or futures?
- How will you ensure that the system is efficient when dealing with large volumes of data?

2. Step 2: Input Parameters from the User

After designing your data handling strategy, the script should prompt the user for the following:

- **Derivative Type:** Which derivative data to work with (e.g., Nifty, BankNifty, FinNifty, BankEx).
- **Timeframes:** Choose multiple timeframes for the backtest (e.g., 1 minute, 5 minutes, 1 hour, 1 day).
- **Expiry Date:** If the segment is options or futures, prompt the user to specify the expiry date (for filtering relevant contracts).
- **Strategy:** Ask the user to choose or define a trading strategy (e.g., Moving Average Crossover, RSI, Bollinger Bands, custom strategies based on signal generation).

3. Step 3: Strategy Implementation and Signal Generation

Implement a **trading strategy** based on the user's input. The strategy should:

- Generate **buy/sell signals** based on predefined rules (e.g., moving averages, RSI, etc.).
- Include logic for **stop-loss** orders to minimize risk.
- Ensure the strategy can dynamically handle the data for the selected timeframe and derivative.

Example Strategies:

- **Moving Average Crossover**:
    - Buy when the short-term moving average crosses above the long-term moving average.
    - Sell when the short-term moving average crosses below the long-term moving average.
    - Implement stop-loss logic to minimize losses.
- **RSI (Relative Strength Index)**:
    - Buy when RSI goes below 30 (indicating oversold).
    - Sell when RSI goes above 70 (indicating overbought).
    - Use stop-loss or trailing stop logic to manage risk.

The strategy should work for **multiple expiries and multiple derivatives**, and the signal generation should take into account different expiry dates for options contracts.

4. Step 4: Backtesting Across Timeframes

- After implementing the strategy, **backtest** it on the resampled data for different timeframes (1 minute, 5 minutes, 1 hour, daily).
- The script should generate results for each timeframe, such as:
    - **Number of trades**.
    - **Total profit/loss**.
    - **Win rate** (percentage of profitable trades).
    - **Risk-adjusted metrics** (Sharpe ratio, maximum drawdown).
- The results should vary based on the timeframe and derivative.

Sample Workflow

1. **Step 1: Data Handling Plan**

    Design your approach to handle large datasets for different derivatives with multiple expiries, including options and futures. Here's an example structure:

    - **Data Storage:** Use a combination of **Parquet** or **Feather** formats to store historical data

   efficiently, partitioned by expiry and derivative (e.g., Nifty, BankNifty).
   - **Data Loading:** Use a chunk-based approach where only the relevant data for a specific derivative and expiry is loaded into memory.
   - **Filtering:** Filter the data dynamically based on the expiry date chosen by the user to ensure that only relevant options contracts or futures data are loaded for backtesting.

2. **Step 2: User Inputs**

   The script prompts the user:

   - **Derivative:** Choose from Nifty, BankNifty, FinNifty, or BankEx.
   - **Expiry:** Input expiry date (if the segment is options or futures).
   - **Timeframes:** Choose timeframes for the backtest.
   - **Strategy:** Choose a predefined or custom trading strategy.

3. **Step 3: Strategy Implementation**

   Implement a **Moving Average Crossover** or **RSI** strategy with **stop-loss** logic, tailored to the chosen derivative and expiry

4. **Step 4: Backtesting and Output**

   Backtest the strategy across **multiple timeframes** and generate output for each

Evaluation Criteria:

1. **Data Management:**

   - Efficient handling of large historical datasets with multiple expiries and different derivatives.
   - A clear strategy for partitioning and filtering data based on expiries and derivatives (e.g., Nifty, BankNifty).

2. **User Input Handling:**

   - Ability to take multiple user inputs (timeframes, expiry dates, strategies) and dynamically adjust the backtesting accordingly.

3. **Strategy Implementation:**

   - Accurate implementation of a trading strategy that generates buy/sell signals and integrates stop-loss logic.
   - Ability to handle different derivatives and expiries in the strategy.

4. **Backtest Results:**

   - The output should include metrics like total profit/loss, number of trades, win rate, and risk-adjusted performance metrics (e.g., Sharpe ratio).
   - Results should differ based on the chosen timeframe and derivative.

5. **Code Quality:**

   - Clean and well-documented code with a focus on modularity, efficiency, and readability.