Task Progression for Strategy Development using backtesting.py

Phase 1: Simple Strategy Development and Backtesting

1. **Task 1: Build a Simple Strategy**
   - Define a basic strategy class using backtesting.py (e.g., moving average crossover).
   - Implement the init method for setting up indicators.
   - Implement the next method for trade logic (e.g., buy when a short-term SMA crosses above a long-term SMA).
2. **Task 2: Backtest the Simple Strategy**
   - Load historical data for a single symbol (e.g., GOOG).
   - Run the backtest using the Backtest class.
   - Analyze key performance metrics (e.g., returns, drawdown, win rate).
3. **Task 3: Optimize the Simple Strategy**
   - Define optimization parameters (e.g., SMA periods).
   - Use the optimize method to find the best-performing parameters.
   - Evaluate the results to ensure parameters are robust.
4. **Task 4: Conduct Walk-Forward Optimization (WFO)**
   - Set up in-sample and out-of-sample periods for WFO.
   - Run WFO to validate strategy robustness across different periods.
   - Review performance consistency and adjust strategy if necessary.

Phase 2: Multi-Symbol Strategy Development

5. **Task 5: Extend Strategy to Multiple Symbols**
   - Modify the strategy class to handle multiple symbols.
   - Update the init and next methods to incorporate data and signals from multiple symbols (e.g., SPY, AAPL, GOOG).
   - Implement logic for trading decisions based on conditions across multiple symbols.
6. **Task 6: Backtest with Multiple Symbols**
   - Load historical data for multiple symbols.
   - Adjust the Backtest setup to handle multiple datasets simultaneously.
   - Execute the backtest and analyze performance for each symbol and the overall strategy.
7. **Task 7: Optimize Multi-Symbol Strategy**

   - Define optimization parameters relevant to each symbol (e.g., different SMAs for each symbol).
   - Run optimization to identify the best parameter combinations for trading multiple symbols.
   - Evaluate cross-symbol performance to ensure the strategy performs well across all symbols.

8. **Task 8: Conduct Walk-Forward Optimization (WFO) with Multiple Symbols**
   - Divide historical data of multiple symbols into sequential in-sample and out-of-sample periods.
   - Run WFO to optimize parameters in each in-sample period and test in the out-of-sample period.
   - Review WFO results for consistency and robustness across multiple symbols.

Phase 3: Multi-Timeframe, Multi-Symbol Strategy Development

9. **Task 9: Incorporate Multiple Timeframes into Strategy**
   - Modify the strategy class to use data from multiple timeframes (e.g., 5-minute, 1-hour, daily).
   - Update the init method to set up indicators across different timeframes.
   - In the next method, use signals from multiple timeframes to make trading decisions (e.g., confirm a trend on a higher timeframe before entering a trade on a lower timeframe).
10. **Task 10: Backtest Multi-Timeframe, Multi-Symbol Strategy**
    - Load historical data for multiple symbols and timeframes.
    - Adjust the Backtest setup to integrate data across symbols and timeframes.
    - Run the backtest, analyzing the combined performance across symbols and timeframes.
11. **Task 11: Optimize Multi-Timeframe, Multi-Symbol Strategy**
    - Define optimization parameters for each symbol and timeframe (e.g., SMA periods, entry thresholds).
    - Run optimization with a focus on finding robust parameters that work across symbols and timeframes.
    - Evaluate the combined performance and robustness of the strategy.
12. **Task 12: Conduct Walk-Forward Optimization (WFO) with Multi-Timeframe, Multi-Symbol Data**
    - Set up WFO by dividing the multi-symbol, multi-timeframe data into in-sample and out-of-sample periods.

  - Perform WFO to optimize in-sample and test out-of-sample for each timeframe and symbol.
  - Review WFO results to ensure the strategy maintains performance consistency across all symbols and timeframes.

This progression guides you from simple strategy development to handling complex multi-symbol, multi-timeframe scenarios using backtesting.py. Let me know if you'd like further details or adjustments!