Below is a high-level, step-by-step explanation of what the code is doing, why it does so, and the overall purpose of computing these quantities.

**1. Overview**

The code is essentially a **backtesting framework** for a trading strategy (or multiple strategies). It reads historical market data from CSV files (either one symbol or two symbols, depending on the strategy) and simulates trades based on certain conditions (e.g., RSI, MACD, Standard Deviation, Linear Regression, Pairs Trading, etc.). Finally, it computes a profit and loss (PnL) figure for the simulated trades over a specified time period.

You can imagine it this way: You have historical price data and a set of “rules” that says, “If X happens, **buy**. If Y happens, **sell**.” The code goes through each day in reverse chronological order (from the most recent day to the oldest in the CSV) and decides whether to buy, sell, or hold. Then it tracks how many shares you own, how much cash you gain or lose each day, and in the end, it closes all positions and reports the final net PnL.

**2. Key Inputs (Command Line Arguments)**

1. **strategy**: which strategy to use
2. **symbol**: the symbol’s name (like ‘AAPL’) for single-stock strategies
3. **n**: a look-back period for certain calculations (e.g., RSI or standard deviation periods)
4. **x**: the maximum number of units (shares, for instance) you can hold long or short
5. **p**: a percentage threshold (or sometimes used as a multiplier threshold)
6. **max\_hold\_days**: the maximum number of days you can hold a position (only used in the DMA++ strategy)
7. **c1, c2**: constants used in the “better DMA” calculations
8. **oversold\_threshold, overbought\_threshold**: used for RSI strategy
9. **adx\_threshold**: used for ADX strategy
10. **threshold**: used for Pairs Trading (z-score threshold)
11. **symbol1, symbol2**: used for pairs trading (the two assets in the pair)
12. **train\_start\_date, train\_end\_date**: the period used for training the **linear regression** model
13. **start\_date, end\_date**: the period on which to actually run the backtest (execute trades)

**3. CSV Parsing**

**3.1 Single-Symbol Strategies**

* The code opens symbol + ".csv" and reads lines in the format:
* Date,Close,High,Low,Open,Trades,VWAP
* It stores these data points in stock\_data and possibly in other helper containers (e.g., for ADX or linear regression).

**3.2 Pairs Trading**

* The code opens symbol1 + ".csv", reads each line (close prices), stores them in a vector.
* The code then opens symbol2 + ".csv", reads each line (close prices), and pairs them with the data from symbol1.
* Essentially, we get parallel time series for two different symbols.

**4. Trading Strategies**

Depending on strategy, the code applies a **different** signal-generation function to decide whether to buy (+1), sell (-1), or do nothing (0):

1. **BASIC** (monotonic\_basic):  
   Checks if the price is monotonically increasing or decreasing over the last n days.
   * If strictly going up for n days, signal = **sell**.
   * If strictly going down for n days, signal = **buy**.
   * Otherwise, do nothing.
2. **DMA** (standard\_deviation):  
   Looks at standard deviation of the price over n+1 days.
   * If the current price is above (mean + p \* std\_dev), sell.
   * If the current price is below (mean - p \* std\_dev), buy.
   * Otherwise, do nothing.
3. **DMA++** (better\_DMA):  
   A variation of the DMA strategy that uses **Adaptive Moving Average** ideas. It also imposes a maximum holding period (max\_hold\_days). Once you open a trade, you can only keep it for so many days.
4. **MACD** + SIGNAL:  
   Computes the MACD line (difference of 12-day and 26-day exponential weighted means) and the Signal line (9-day exponential weighted average of MACD).
   * If MACD > Signal line, buy.
   * If MACD < Signal line, sell.
5. **RSI**:  
   Uses the **Relative Strength Index** over n days.
   * If RSI < oversold\_threshold, buy.
   * If RSI > overbought\_threshold, sell.
6. **LINEAR\_REGRESSION**:
   * Trains a linear regression model using data from train\_start\_date to train\_end\_date.
   * The features might include Open, VWAP, High, Low, Trades, etc.
   * Once it trains the model (via a **Normal Equation** solver), it uses those parameters to predict tomorrow’s closing price.
   * If the predicted price is sufficiently higher (by more than p%), buy. If it’s sufficiently lower (by more than p%), sell.
7. **ADX**:
   * Computes the **Average Directional Index** to detect trend strength.
   * If ADX > adx\_threshold, buy signal.
   * If ADX < adx\_threshold, sell signal.  
     (Implementation details: computing +DI, -DI, ATR, then DX, and smoothing into ADX.)
8. **PAIRS**:
   * For two symbols (symbol1 and symbol2), compute their **spread** (difference in their prices).
   * Calculate a rolling mean and standard deviation of the spread over the last n days.
   * If the current spread’s z-score is above threshold, you **sell** the first and **buy** the second (betting the spread will revert).
   * If the current spread’s z-score is below -threshold, you **buy** the first and **sell** the second.

**5. Signal to Trades**

Once a **signal** (buy/sell/do nothing) is generated on a given day:

* The code checks if it violates the maximum position size (x) in either direction (long or short).
* If not, it executes that trade:
  + **Buy**: cash\_flow decreases by the price, and shares increase by 1.
  + **Sell**: cash\_flow increases by the price, and shares decrease by 1.

In the **PAIRS** strategy, it tracks how many units of each symbol it holds (no\_of\_stocks1 and no\_of\_stocks2) separately and places offsetting trades on the two symbols if the spread calls for it.

In **DMA++**, it uses a queue to remember the day a position was opened so it can forcibly close it after max\_hold\_days.

**6. Outputs / Results**

1. **daily\_cashflow.csv**  
   Records Date, Cashflow on each iteration/day, showing how total cash changes daily based on trades.
2. **order\_statistics.csv** (or order\_statistics\_1.csv and order\_statistics\_2.csv for pairs)  
   Lists each **order** placed, with columns: Date, Order\_dir, Quantity, Price.
3. **final\_pnl.txt**  
   The final net profit or loss after all positions are closed on the last day.

**7. Why Find These Quantities?**

1. **Backtesting**: The primary goal is to see how a trading strategy would have performed historically. By simulating buys and sells on past data, you can measure performance, risk, and other metrics *before* trading with real money.
2. **Parameter Tuning and Strategy Comparison**:
   * The code can help you compare different strategies (DMA, RSI, MACD, etc.) on the same historical data.
   * You can also tweak n, p, oversold\_threshold, overbought\_threshold, etc. to see which parameters yield the best backtest results.
3. **Proof of Concept**: It can serve as a blueprint for building a more advanced trading system (e.g., hooking it into real-time data, applying risk management, etc.).
4. **Model Explanation / Interpretation**:
   * The **Linear Regression** example shows how you might incorporate machine learning models.
   * The code also demonstrates how standard indicators like MACD, RSI, ADX, etc., can be coded and integrated into a strategy.

In short, each portion of the code—reading data, computing indicators, generating buy/sell signals, tracking daily PnL, and writing the final results—is part of a *backtest pipeline* to determine whether a given strategy is profitable under certain historical conditions.