Autoregressive Integrated Moving Average (ARIMA)

Logic:

The ARIMA model is a statistical method that forecasts future prices based on their own past values. It assumes that the future price is a linear combination of past prices and past forecast errors. The model has three components:

- **Autoregressive (AR):** The price is regressed on its own previous values.
- **Integrated (I):** The price data is differenced to make it stationary (i.e., to remove trends and stabilize its mean and variance).
- **Moving Average (MA):** The forecast error is modeled as a combination of past error terms.

Formula:

The ARIMA (p, d, q) model is expressed as:

$$\varphi(B)(1 - B)^d y_t = \theta(B)\epsilon_t$$

where:

- y_t is the price at time t.
- d is the degree of differencing.

- $\varphi(B)$ is the autoregressive operator.
- $\theta(B)$ is the moving average operator.
- ϵ_t is the white noise error term.

Support Vector Regression (SVR)

Logic:
SVR is a machine learning technique that finds a function that best fits the data by creating a "margin of tolerance" around the predicted values. Any errors that fall within this margin are ignored, which makes the model robust to noise and outliers. To handle non-linear price movements, SVR uses a **kernel function** to map the data into a higher-dimensional space where a linear relationship can be found.

Formula:
The SVR model is represented by the function:

$$f(x, \alpha, \alpha^*) = \sum_{i=1}^{N} (\alpha_i - \alpha^*_i) \phi(x, x_i) + b$$

where:

- $f(x, \alpha, \alpha^*)$ is the predicted price.
- $\phi(x, x_i)$ is the kernel function.
- α_i and α^*_i are Lagrange multipliers that define the support vectors.
- b is the bias term.

Multilayer Perceptron (MLP)

Logic:
An MLP is a type of artificial neural network with an input

layer, one or more hidden layers, and an output layer. It learns complex, non-linear relationships in price data by adjusting the **weights** between its neurons during training. This process, known as **backpropagation**, aims to minimize the error between the model's predicted prices and the actual prices.

Formula:
The output of a single neuron in an MLP is given by:

$$Y = \sigma(\sum_{i=1}^{N} w_i x_i + b)$$

where:

- Y is the neuron's output.
- x_i are the inputs.
- w_i are the weights.
- b is the bias.
- σ is a non-linear activation function (like ReLU or sigmoid).

Recurrent Neural Network (RNN)

Logic:
RNNs are specifically designed for sequential data like time-series prices. Unlike MLPs, an RNN has a "memory" or **hidden state** that retains information from previous time steps. The output from the previous step is fed as an input to the current step, allowing the network to capture temporal patterns and dependencies in the price history.

Formula:
The core computations in an RNN are:

$$h_{t+1} = f_H(W_U \cdot x_t + W_V \cdot h_t)$$

$$o_{t+1} = f_O(W_W \cdot h_{t+1})$$

where:

- h_t is the hidden state at time t.
- x_t is the input at time t.
- o_{t+1} is the output (predicted price) at time t+1.
- W_U, W_V, W_W are the weight matrices.
- f_H, f_O are activation functions.

Long Short-Term Memory (LSTM)

Logic:
LSTM is an advanced type of RNN that can learn long-term dependencies in price data by overcoming the limitations of traditional RNNs. It uses a unique architecture with three "gates":

- **Forget Gate:** Decides what information to discard from the cell state.
- **Input Gate:** Determines what new information to store in the cell state.
- **Output Gate:** Controls what information to pass on to the next hidden state.
  This gating mechanism allows LSTM to effectively remember important patterns over long periods while ignoring irrelevant noise, making it highly suitable for volatile commodity prices.

Formula:
An LSTM cell is defined by the following equations:

- **Forget Gate:** $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
- **Input Gate:** $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
- **Candidate Cell State:** $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
- **Cell State Update:** $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$
- **Output Gate:** $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
- **Hidden State:** $h_t = o_t \cdot \tanh(C_t)$

Gated Recurrent Unit (GRU)

Logic:
A GRU is a simplified version of the LSTM that also captures long-term dependencies but with a more computationally efficient architecture. It combines the forget and input gates of an LSTM into a single **update gate** and also has a **reset gate**.

- **Update Gate:** Decides how much of the past information to carry forward.
- **Reset Gate:** Determines how to combine the new input with the previous memory.

Formula:
A GRU cell is defined by the following equations:

- **Update Gate:** $z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$
- **Reset Gate:** $r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$
- **Candidate Hidden State:** $\tilde{h}_t = \tanh(W \cdot [r_t \odot h_{t-1}, x_t] + b)$
- **Hidden State:** $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$

Extreme Gradient Boosting (XGBoost)

Logic:
XGBoost is an ensemble learning algorithm that builds a series of decision trees sequentially. Each new tree is trained to correct the errors made by the previous set of trees. It enhances standard gradient boosting with regularization to prevent overfitting and parallel processing for efficiency. It models non-linear patterns by progressively refining its predictions.

Formula:
The core idea is to iteratively add new functions (trees) that improve the prediction. The prediction at step t is updated as: $$\hat{y}_{i}^{(t)} = \hat{y}_{i}^{(t-1)} + f_{t}(x_{i})$$The model minimizes an objective function that includes both the training loss and a regularization term to control complexity:

$$L(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Echo State Network (ESN)

Logic:
An ESN is a type of RNN with a very efficient training method. It consists of a large, fixed, and randomly generated internal recurrent layer called a **"reservoir"**. The key idea is that only the connections from the reservoir to the output layer are trained; the internal connections remain unchanged. The reservoir transforms the input price series into a higher-dimensional space of dynamic patterns, and the output layer simply learns a linear mapping from these patterns to the final price prediction.

Formula:
The ESN is described by two main equations:

- **Reservoir State Update:** $x_t = \tanh(W_{in} \cdot u_t + W \cdot x_{t-1})$
- **Output:** $y_t = W_{out} \cdot x_t$
  where:
- x_t is the state of the reservoir.
- u_t is the input price.
- W_{in} and W are the fixed input and reservoir weight matrices.

- W_{out} is the trainable output weight matrix.