1. Stock Price Function Generation:

To simulate realistic stock price data, I created a custom function that generates stock prices for a period of 5900 days. This function incorporates various real-life factors to mimic real market behavior, ensuring the generated data reflects realistic stock price trends.

Key elements of the stock price generation function:
- **Random Trend:** A gradual upward trend is applied to simulate how stock prices tend to increase over time.
- **Noise and Volatility**: I introduced random noise to the data to capture the inherent volatility of stock markets. This noise mimics unpredictable market behavior, ensuring that prices fluctuate in a non-linear manner.
- **Weekly Trend**: To reflect the different behavior of stock prices on weekdays versus weekends, the function accounts for the reduced activity and price movement over weekends, simulating lower volatility during non-trading days.
- **Sinusoidal Oscillations**: Daily stock price fluctuations follow a sinusoidal pattern to simulate the natural rise and fall of stock prices within a day.

Features Generated:
- **Opening_Price**: The price at the start of each trading day.
- **Closing_Price**: The price at the end of each day.
- **Highest_Price and Lowest_Price**: The highest and lowest prices observed during the day.
- **Average_Price**:The average of the highest and lowest prices for the day, giving a good indicator of the day’s overall price movement.

Technical Indicators:
- **RSI** (Relative Strength Index): A momentum indicator that measures the magnitude of recent price changes to evaluate overbought or oversold conditions in the market.
- **SMA7** (Simple Moving Average, 7 days): The 7-day simple moving average smooths out short-term fluctuations to reveal the direction of the price trend.
- **SMA30** (Simple Moving Average, 30 days):The 30-day simple moving average represents a longer-term trend compared to the 7-day SMA.
- **EMA7** (Exponential Moving Average, 7 days): This indicator places greater weight on recent prices, making it more responsive to new information compared to SMA.

2. Data Preprocessing:

After generating stock price data for 5900 days, I created a DataFrame that includes all the features mentioned above. This DataFrame serves as the foundation for further preprocessing before model training.

- Feature Engineering: I added a new feature, `**Prev_Closing_Price**`, which stores the closing price of the previous day, capturing sequential dependencies crucial for time series modeling.
- **Standardization**: I applied Standscaler to all the features. This is essential for ensuring that the LSTM model handles the features uniformly and avoids bias caused by varying scales of the input data.
- **Time Step Window** (30 Days): Given the dataset spans 5900 days, I used a 30-day time step for the model input, meaning that the prediction for the next day is based on the previous 30 days of data. This ensures the model learns patterns from recent history to make future predictions.

3. LSTM Model Architecture:

For predicting the Closing_Price, I designed a two-layer LSTM model, as LSTM is particularly well-suited for time series forecasting. Here's how the architecture works:

- **LSTM Layers**: I used two LSTM layers that process the sequential nature of the stock price data. LSTMs are effective because they retain important information from earlier time steps, which is crucial for stock price prediction.
- **Dropout Regularization**:Between the LSTM layers, I implemented dropout regularization to mitigate overfitting by randomly ignoring a fraction of neurons during training, ensuring the model generalizes well to unseen data.
- **Training and Validation**: After tuning the model, I proceeded with training, adjusting hyperparameters to achieve acceptable training and validation loss, ensuring the model could make accurate predictions without overfitting.

4. Anomaly Detection

After building the predictive LSTM model, I implemented two different approaches for detecting anomalies in stock prices:

Approach 1: Threshold-Based Anomaly Detection

- **Reconstruction of Training Data**: After training the model, I reconstructed the training data using the predicted values.
- **Threshold Calculation**: I calculated the mean and standard deviation of the reconstruction error (i.e., the difference between actual and predicted closing prices). The threshold for anomalies was set as a function of the mean and standard deviation.
- Anomaly Highlighting:Data points where the reconstruction error exceeded the threshold were classified as anomalies. I visualized these anomalies by plotting the actual vs. predicted prices,

highlighting deviations where the model's predictions differed significantly from the actual values.

Approach 2: Isolation Forest-Based Anomaly Detection

- **Isolation Forest Algorithm**: The second method involves using the Isolation Forest algorithm, which identifies anomalies by isolating data points that are far from the norm. I used the following formula to implement the isolation forest:

  - The **contamination** parameter (set to **0**.**05**) specifies the proportion of anomalies in the dataset, while the random_state ensures reproducibility. The isolation forest works by randomly selecting features and splitting data points, isolating those that behave differently from the majority.
- Anomaly Detection: After fitting the isolation forest model to the reconstruction error, I identified the outliers (i.e., anomalies) and visualized them along with the stock price predictions, marking points where the model detected abnormal behavior.