**Project Report**

1. **Data Loading and Exploration:**

- Loaded a CSV file named 'all\_stocks\_5yr.csv' into a DataFrame (`df`).

- Checked the number of unique values in the 'Name' column, which suggests there might be multiple stocks in the dataset.

- Converted the 'date' column to datetime format.

- Printed dataset description (summary statistics) to understand the data distribution.

- Printed the range of dates covered in the dataset.

- Plotted the daily closing prices over time to visualize the data.

2. **Data Cleaning:**

- Checked for missing values and printed the count of missing values before and after filling them.

- Filled missing values with the mean of each respective column.

- Normalized and standardized the selected columns ('open', 'high', 'low', 'close', 'volume').

3. **Visualization:**

- Plotted each column as subplots to visualize their distributions over time.

4. **Time Series Analysis:**

- Utilized the matplotlib library to plot the time series data and visually inspect whether the time series is stationary or not.

5. **Modeling Techniques:**

- Splitting the data into training and testing sets.

- Selecting appropriate features for modeling.

- Training different models (e.g., SVR, ARIMA, LSTM) on the training data.

- Evaluating the models using appropriate metrics (e.g., Mean Squared Error for regression models).

- Fine-tuning hyperparameters using techniques like GridSearchCV for better model performance.

- Selecting the best-performing model for forecasting.

**ARIMA MODEL:**

1. **Data Preparation and Visualization:**
   * I have the 'close' price column for time series forecasting.
   * Plotted the original time series data to visualize the trend and seasonality.
2. **Stationarity Check:**
   * Conducted Augmented Dickey-Fuller (ADF) test to check stationarity.
   * Calculated and plotted rolling mean and standard deviation to visualize any trends or seasonality in the data.
   * Differenced the series to make it stationary and performed ADF test on the differenced series.
3. **Autocorrelation Analysis:**
   * Plotted Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) to determine the values of AR and MA terms for the ARIMA model.
4. **Model Configuration:**
   * Based on the ACF and PACF plots, you selected AR (p), differencing (d), and MA (q) terms as 1.
5. **Data Splitting:**
   * Split the data into train and test sets (80% training, 20% testing), ensuring no shuffling to maintain the time order.
6. **Model Fitting and Forecasting:**
   * Utilized the ARIMA model from **statsmodels** library and fit it to the training data.
   * Forecasted future values using the trained model.
   * Generated predictions for the entire test set.
7. **Model Evaluation:**
   * Plotted the original time series along with the predicted values to visually compare the forecasted values against the actuals

print(f'Accuracy: {accuracy \* 100:.2f}%')  
Accuracy: 56.93%

**ANN Design and Training:**

1. **Data Preparation:**
   * I selected the 'close' price column for time series forecasting.
   * Scaled the dataset using MinMaxScaler to ensure all features are within the range [0, 1].
2. **Dataset Creation for ANN:**
   * Created a function to create the dataset for the ANN model with a look-back window of 3 days.
   * Split the data into input features (X) and target variable (Y) based on the specified look-back window.
3. **Data Splitting:**
   * Split the dataset into training and testing sets with an 80:20 ratio.
4. **Reshaping Input Data:**
   * Reshaped the input data to match the expected input shape for the ANN model.
5. **Model Architecture:**
   * Designed a Sequential ANN model with three dense layers.
   * Added dropout layers to prevent overfitting.
   * Compiled the model with Mean Squared Error (MSE) loss function and Adam optimizer.
6. **Model Training:**
   * Trained the ANN model on the training data for 30 epochs with a batch size of 10.
   * Used 20% of the training data as validation data to monitor model performance during training.
7. **Model Evaluation:**
   * Evaluated the model on the test data and calculated the loss and accuracy.
   * Inverse scaled the predictions to obtain the original scale.
   * Calculated additional metrics including precision, recall, F1-score, and accuracy using scikit-learn's metrics functions.
8. **Plotting Predictions vs Actuals:**
   * Plotted the actual 'close' prices against the predicted prices to visualize the model's performance.

# SARIMA (Seasonal ARIMA)

1. **Data Preparation:**
   * I focused on the 'close' price column for time series forecasting.
2. **Seasonal Decomposition:**
   * Checked if the dataset has at least 2 years of data (24 months) for monthly decomposition.
   * If the condition is met, performed seasonal decomposition using the **seasonal\_decompose** function with an additive model and a seasonal period of 12 (assuming monthly data).
   * Plotted the decomposed components including trend, seasonal, and residual.
3. **Data Splitting:**
   * Split the time series data into training and testing sets using an 80:20 ratio.
4. **SARIMA Model Configuration:**
   * Configured the SARIMA model with order=(1, 1, 1) for the non-seasonal part and seasonal\_order=(1, 1, 1, 12) for the seasonal part.
   * Disabled stationarity and invertibility enforcement.
5. **Model Fitting and Forecasting:**
   * Fitted the SARIMA model to the training data.
   * Forecasted future values using the trained SARIMA model.
6. **Metrics Calculation:**
   * Calculated the mean of the test data and the predicted values.
   * Converted the test and predicted values into binary classes based on whether they are greater than the mean.
   * Computed precision, recall, and F1-score to evaluate the accuracy of the model's binary classification.
7. **Plotting Forecast vs Actuals:**
   * Plotted the actual 'close' prices against the forecasted prices to visually compare the model's performance.

**Exponential Smoothing (ETS)**

1. **Data Preparation:**
   * Selected the 'close' prices column for time series forecasting.
2. **Data Splitting:**
   * Split the time series data into training and testing sets using an 80:20 ratio.
3. **Model Fitting:**
   * Utilized the Triple Exponential Smoothing (Holt-Winters) model for forecasting.
   * Configured the model with additive trend, additive seasonal components, and a damped trend.
   * Fit the model to the training data, with seasonal periods set to 365 assuming daily data.
4. **Forecasting:**
   * Generated forecasts for the length of the test set using the trained ETS model.
5. **Plotting Forecast vs Actuals:**
   * Plotted the actual 'close' prices of the training and testing sets.
   * Superimposed the forecasted values onto the plot for comparison.
6. **Classification Metrics Calculation:**
   * Classified the forecasted values and actual test data into binary outcomes based on whether the predictions are above or below the mean of the training data.
   * Calculated precision, recall, F1-score, and accuracy as classification metrics to evaluate the model's performance.

The Exponential Smoothing model demonstrates a reasonable ability to capture the trend and seasonality in the time series data.

# Prophet

**Data Preparation:**

* Prepared the DataFrame for Prophet by resetting the index and selecting only the 'date' and 'close' columns. Renamed the columns to 'ds' and 'y' respectively to fit Prophet's requirements.

**Data Splitting:**

* Split the prepared DataFrame into training and testing sets with an 80:20 ratio.

**Model Initialization and Fitting:**

* Initialized the Prophet model with daily, yearly, and weekly seasonality.
* Fit the model to the training data.

**Forecasting:**

* Created a future DataFrame for predictions with the same frequency as the original data (daily).
* Generated predictions using the fitted model.

**Plotting:**

* Plotted the forecasted values along with the historical data using Prophet's built-in plotting capabilities.
* Additionally, plotted the components of the forecast (trend, yearly seasonality, and weekly seasonality) to visualize the decomposition of the time series.

**Metrics Calculation:**

* Filtered the forecast to include only the test data range.
* Converted both the predictions and actuals to binary outcomes based on whether they are above or below the mean of the training data.
* Calculated precision, recall, F1-score, and accuracy to evaluate the binary classification performance of the model.

The Prophet model appears to capture the trend and seasonality of the time series data effectively.

# SVR

**Feature Selection and Preparation:**

* Selected the 'open', 'high', 'low', and 'volume' columns as independent variables (features) and 'close' column as the dependent variable (target).
* Standardized the independent variables using StandardScaler.

**Data Splitting:**

* Split the standardized data into training and testing sets with a test size of 20%.

**Parameter Tuning and Model Selection:**

* Utilized GridSearchCV to perform parameter tuning and model selection for SVR.
* Tuned parameters included the kernel type ('rbf', 'linear', 'poly'), the regularization parameter (C), and the kernel coefficient (gamma).
* The scoring metric used for optimization was negative mean squared error (neg\_mean\_squared\_error).

**Best Model Selection:**

* Selected the best SVR model based on the tuned parameters from GridSearchCV.

**Prediction:**

* Predicted the target variable ('close' prices) for the test set using the best SVR model.

**Plotting Actual vs Predicted Values:**

* Plotted the actual 'close' prices against the predicted prices to visually compare the model's performance.

**Classification Metrics Calculation:**

* Converted both the actual and predicted 'close' prices to binary classes based on whether they are above or below the mean of the training data.
* Calculated precision, recall, F1-score, and accuracy to evaluate the binary classification performance of the model.

The SVR model demonstrates a reasonable ability to predict 'close' prices based on the selected features

# Long Short-Term Memory (LSTM)

**Data Preparation:**

* Selected the 'close' prices column and converted it to float data type for training.
* Scaled the data using MinMaxScaler.

**Sequence Creation for Training:**

* Created sequences of data for training with a time step of 100.
* Reshaped the data into input-output pairs where X=t, t+1, t+2,..., t+99 and Y=t+100.

**Data Splitting:**

* Split the dataset into training and testing sets with an 80:20 ratio.

**Model Architecture:**

* Created an LSTM model with two LSTM layers with 50 units each.
* Added a Dense output layer with one unit.
* Compiled the model using the Adam optimizer with a learning rate of 0.01 and mean squared error loss function.

**Model Training:**

* Trained the model on the training data for 10 epochs with a batch size of 64.
* Used the testing data as validation data during training.

**Prediction:**

* Predicted the 'close' prices for both training and testing data using the trained LSTM model.
* Inverse scaled the predictions to obtain the original scale.

**Plotting:**

* Plotted the true 'close' prices against the LSTM predicted values for the testing data to visualize the model's performance.

**Classification Metrics Calculation:**

* Converted both the actual and predicted 'close' prices to binary classes based on whether they are above or below the mean of the training data.
* Calculated precision, recall, F1-score, and accuracy to evaluate the binary classification performance of the model.

The LSTM model appears to capture the underlying patterns and trends in the time series data effectively

# Hybrid Models Integration

**Data Preparation:**

* Selected the 'close' prices column and converted it to float data type for analysis.

**Data Splitting for ARIMA:**

* Split the data into training and testing sets with an 80:20 ratio for the ARIMA model.

**ARIMA Model:**

* Configured and fitted an ARIMA model with an order of (1, 1, 1) using the training data.

**ARIMA Prediction:**

* Forecasted future values using the ARIMA model to obtain residuals.

**Scaling Residuals for ANN:**

* Scaled the residuals obtained from the ARIMA prediction using MinMaxScaler to prepare for input to the ANN model.

**ANN Model for Residuals:**

* Created and trained an ANN model with one hidden layer containing 50 units and ReLU activation function, predicting the scaled residuals.

**ANN Prediction:**

* Used the trained ANN model to predict the scaled residuals.

**Conversion to Original Scale and Combined Prediction:**

* Inverse scaled the ANN predictions to obtain the residuals in the original scale.
* Combined the ARIMA predictions and the ANN-predicted residuals to get the final prediction.

**Plotting Results:**

* Plotted the actual 'close' prices against the hybrid ARIMA + ANN predicted prices for the testing data to visualize the model's performance.

**Metrics Calculation - Binary Classification:**

* Classified both the actual and predicted 'close' prices into binary outcomes based on whether they are above or below the mean price.
* Calculated precision, recall, F1-score, and accuracy to evaluate the binary classification performance of the model.

The hybrid ARIMA + ANN model demonstrates an effective combination of time series forecasting with machine learning techniques. The model captures the underlying patterns and trends in the time series data and provides reasonably accurate predictions