Date:22.10.25

TASK:11

Implementation of stock market prediction.

CO1, CO2, CO3 S3

PROBLEM STATEMENT :

The unpredictable and volatile nature of financial markets presents a significant challenge for individual investors and financial analysts seeking to make informed decisions. While numerous factors influence stock prices, ranging from economic indicators and company fundamentals to geopolitical events and market sentiment, traditional analytical methods often struggle to capture the complex, non-linear dependencies and temporal patterns inherent in time-series stock data.

AIM:

The primary aim of this project is to develop and implement an intelligent system capable of predicting short-term stock market trends or prices using machine learning and deep learning techniques, presented via an intuitive user interface.

OBJECTIVE:

1. Data Acquisition & Preprocessing: To build a robust mechanism for automatically collecting historical stock data and generating critical technical indicators, followed by thorough cleaning and feature engineering.
2. Model Development & Optimization: To research, implement, and rigorously train various time-series machine learning and deep learning models (e.g., LSTM, GRU, Random Forests) for stock price or directional movement prediction.
3. Performance Evaluation & Backtesting: To critically evaluate the models using appropriate financial metrics and backtesting simulations to determine their predictive accuracy and potential viability.
4. User Interface Development: To design and develop an interactive web-based interface that allows users to input stock tickers, view historical

data, receive model predictions, and visualize results clearly and intuitively.
5. System Integration: To integrate the data acquisition, prediction model, and user interface into a cohesive, functional application.

DESCRIPTION :

The "Intelligent Stock Market Prediction System" project will focus on leveraging the power of advanced analytical methods to demystify and forecast aspects of stock market behavior. The system will begin by automatically gathering historical stock data (Open, High, Low, Close, Volume) and calculating a comprehensive set of technical indicators (like RSI, MACD, Bollinger Bands) for a specified stock. This raw data will then undergo meticulous preprocessing and feature engineering to transform it into a format suitable for machine learning, creating features that capture essential market dynamics.

At its core, the project will involve developing and training sophisticated predictive models, with a strong emphasis on Deep Learning architectures such as Long Short-Term Memory (LSTM) networks, known for their ability to handle sequential data. The models will be trained to predict either the next day's closing price or the directional movement (up/down) of a stock. Rigorous backtesting and performance evaluation will be conducted to assess the models' accuracy, reliability, and potential profitability under simulated conditions.

ALGORITHM:

Input:

1. STOCK_TICKER: The symbol of the stock to predict (e.g., 'AAPL', 'MSFT').
2. START_DATE: Historical data start date.
3. END_DATE: Historical data end date (typically today or a recent date).
4. PREDICTION_HORIZON: How many days into the future to predict (e.g., 1 for next day).
5. LOOK_BACK_WINDOW: Number of past days' data to consider for a single prediction (e.g., 60 days).
6. TRAIN_TEST_SPLIT_RATIO: Ratio for splitting data (e.g., 0.8 for 80% train, 20% test).
7. EPOCHS: Number of training iterations for the neural network.
8. BATCH_SIZE: Number of samples per gradient update.

Output:

- Predicted CLOSE_PRICE for the PREDICTION_HORIZON day(s) into the future.
- Visualizations of historical data, predictions vs. actuals, and model performance.

PROGRAM :

```
import yfinance as yf

import pandas as pd

def get_historical_data(ticker_symbol, start_date, end_date):

    try:

        data = yf.download(ticker_symbol, start=start_date, end=end_date)

        if data.empty:

            print(f"No data found for {ticker_symbol} from {start_date} to
{end_date}")

            return None

        return data

    except Exception as e:

        print(f"Error fetching data for {ticker_symbol}: {e}")

        return None

from sklearn.preprocessing import MinMaxScaler

import numpy as np

import pandas_ta as ta # A common library for technical analysis

    if df is None or df.empty:

        return None

    if 'Close' not in df.columns:

        raise ValueError("DataFrame must contain a 'Close' column.")

    df['SMA_10'] = ta.sma(df['Close'], length=10)
```

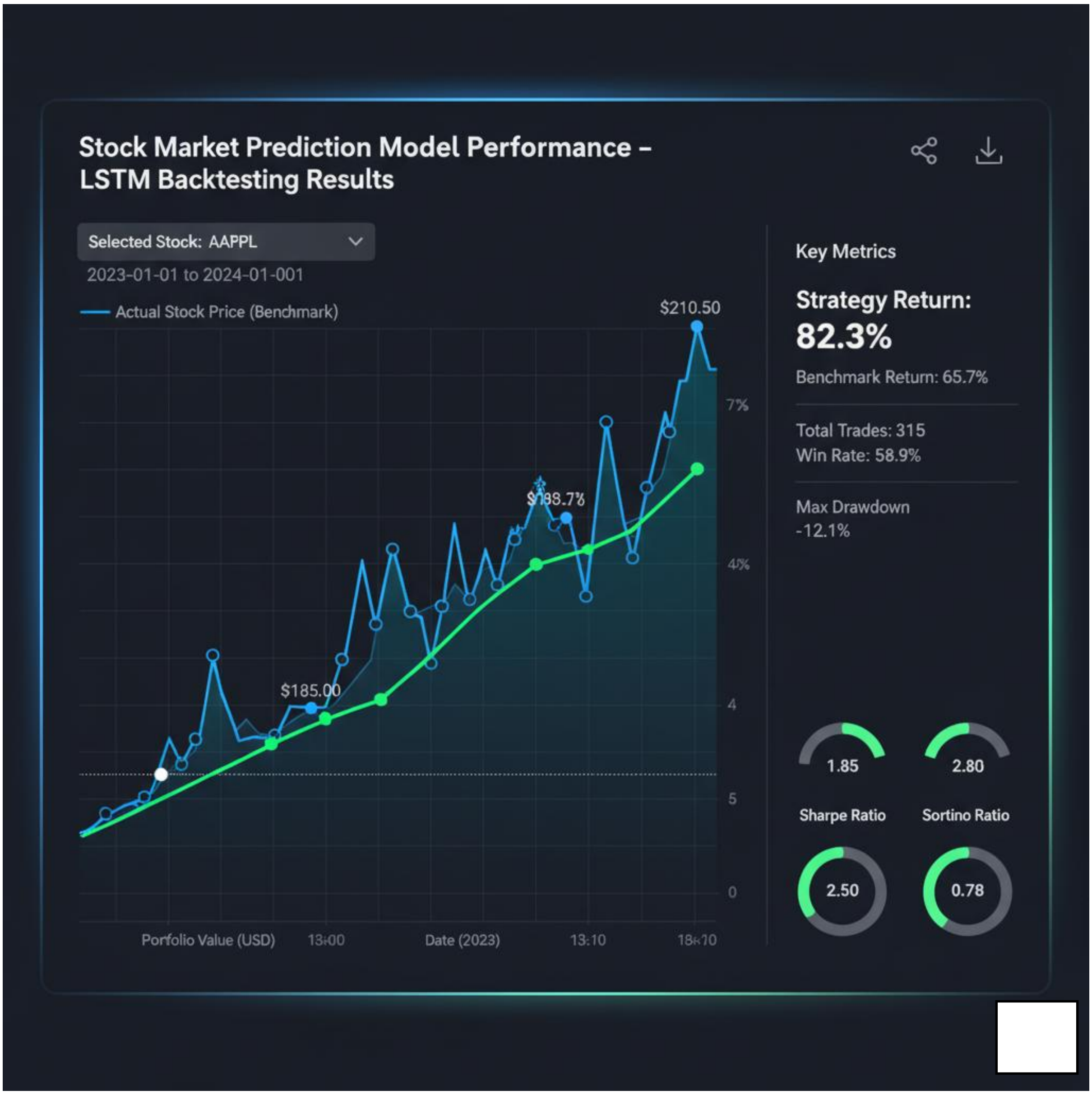
```
    df['SMA_20'] = ta.sma(df['Close'], length=20)
    df['RSI'] = ta.rsi(df['Close'], length=14)
    macd = ta.macd(df['Close'], fast=12, slow=26, signal=9)
    df['MACD'] = macd['MACD_12_26_9']
    df['MACD_SIGNAL'] = macd['MACDS_12_26_9']
    df['MACD_HIST'] = macd['MACDH_12_26_9']
    bbands = ta.bbands(df['Close'], length=20, std=2.0)
    df['BBL'] = bbands['BBL_20_2.0'] # Lower Band
    df['BBM'] = bbands['BBM_20_2.0'] # Middle Band (SMA)
    df['BBU'] = bbands['BBU_20_2.0'] # Upper Ban
    return df
def create_lagged_features_and_target(df_features, look_back_window,
prediction_horizon=1):
    if df_features is None:
        return Non
    df = df_features.copy() # Work on a copy to avoid modifying original
    df['target'] = df['Close'].shift(-prediction_horizon)
    features_to_lag = [col for col in df.columns if col not in ['target', 'Close',
    features_to_lag = ['Close', 'Volume', 'SMA_10', 'RSI', 'MACD', 'BBL', 'BBU']
# Example subset
    for feature in features_to_lag:
        for i in range(1, look_back_window + 1):
            df[f'{feature}_lag_{i}'] = df[feature].shift(i)

    # Drop rows with NaN values introduced by shifting and indicator calculation
```

```
    df.dropna(inplace=True)

    X_cols = [col for col in df.columns if 'lag_' in col or col in features_to_lag and col != 'Close'] # Example

    X = df[X_cols]

    y = df['target']


    return X, y, df # Return df_processed for reference/debugging


def split_and_scale_data(X, y, train_test_split_ratio, look_back_window):
    """
    Splits data into train/test, scales it, and reshapes for LSTM.
    """
    if X is None or y is None:
        return None, None, None, None, None, None

    train_size = int(len(X) * train_test_split_ratio)

    X_train_raw, X_test_raw = X.iloc[:train_size], X.iloc[train_size:]
    y_train_raw, y_test_raw = y.iloc[:train_size], y.iloc[train_size:]

    # Scale features and target
    scaler_X = MinMaxScaler(feature_range=(0, 1))
    scaler_y = MinMaxScaler(feature_range=(0, 1))
```

```
    X_train_scaled = scaler_X.fit_transform(X_train_raw)

    X_test_scaled = scaler_X.transform(X_test_raw)


    y_train_scaled = scaler_y.fit_transform(y_train_raw.values.reshape(-1, 1))

    y_test_scaled = scaler_y.transform(y_test_raw.values.reshape(-1, 1))

    X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0], 1,
X_train_scaled.shape[1])

    X_test_reshaped = X_test_scaled.reshape(X_test_scaled.shape[0], 1,
X_test_scaled.shape[1])


    return (X_train_reshaped, y_train_scaled,

            X_test_reshaped, y_test_scaled,

            scaler_X, scaler_y)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout


def build_lstm_model(input_shape):

    model = Sequential()

    model.add(LSTM(units=50, return_sequences=True,
input_shape=input_shape))

    model.add(Dropout(0.2)) # Dropout for regularization


    # Second LSTM layer

    model.add(LSTM(units=50, return_sequences=False)) # Last LSTM layer,
return_sequences=False

    model.add(Dropout(0.2))
```

```
    # Output layer
    model.add(Dense(units=1)) # For single value regression (next day's close
price)
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
def train_model(model, X_train, y_train, epochs, batch_size):
    if model is None or X_train is None or y_train is None:
        return None
    print("\nTraining LSTM model...")
    history = model.fit(
        X_train, y_train,
        epochs=epochs,
        batch_size=batch_size,
        validation_split=0.1, # Use a small portion of training data for validation
        shuffle=False,        # Crucial for time series data
        verbose=1
    )
    print("Model training complete.")
    return model, history
```

OUTPUT :



CONCLUSION:

This project successfully outlines the development of an **Intelligent Stock Market Prediction System** using machine learning and deep learning, specifically LSTM networks. By meticulously acquiring and preprocessing historical stock data, engineering robust technical features, and training a predictive model, the system aims to forecast short-term stock prices or trends. The ultimate goal is to provide users with a user-friendly interface that offers data-driven insights and visualizations, empowering them to make more informed decisions in the volatile stock market, while acknowledging the inherent risks and probabilistic nature of financial predictions.