STOCK PRICE PREDICTION

Minor Project

- P Jasmitha Sai

Introduction:

In this work, the influence of Random Forest Regression and Long Short-Term Memory (LSTM) machine learning techniques on stock price prediction are examined. Because of the volatility of the market and the actions of investors, predicting stock prices is difficult. Nonlinearities are a challenge for traditional procedures, which is why innovative techniques are being adopted.
Both models are trained and assessed using historical stock data, including daily open, high, low, close prices, and volume, in addition to artificial attributes such closing prices from the prior day. While LSTM is excellent at capturing temporal dependencies, Random Forest Regression provides strong ensemble learning.
This works compares the efficacy of each approach using performance metrics including Mean Squared Error (MSE) and Mean Absolute Error (MAE). By doing so, it provides financial decision-makers with insights into how machine learning might be used in stock market forecasting.

Abstract:

Predicting stock prices is still a difficult but important endeavor in financial markets. This abstract investigates the use of LSTM (Long Short-Term Memory) and Random Forest Regression as machine learning approaches for stock price prediction. We used historical market data, trade volume, derived features, and daily open, high, low, and closing prices. Metrics like Mean Absolute Error (MAE) and Mean Squared Error (MSE) were used to train and assess both models. The outcomes demonstrate how well these methods capture intricate patterns and increase prediction accuracy, providing insightful information to stakeholders who are considering using cutting-edge machine learning techniques for financial forecasting.

WHY RandomForest and LSTM?

Random Forest and LSTM beat Linear Regression for stock price prediction owing to their ability to handle nonlinear correlations (Random Forest) and temporal dependencies (LSTM) in financial time series data. Random Forest uses several decision trees to account for feature interactions and nonlinearities, whereas LSTM's architecture enables it to learn and recall patterns over time, which is critical for capturing stock price trends. These models improve predictive accuracy by utilizing advanced machine learning approaches that may reveal complex patterns and relationships found in dynamic market data, giving useful insights for financial forecasting and decision-making.

Explanation of the Code:

Step 1: Load the Excel file

```python
data = pd.read_excel('/content/dataset.xlsx.xlsx')
```

This line loads an Excel file named dataset.xlsx.xlsx using pandas read_excel function and assigns it to a DataFrame named data.

Step 2: Data Preprocessing

```
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)
```

Converts the 'Date' column to datetime format using pd.to_datetime.

- Sets the 'Date' column as the index of the DataFrame using set_index('Date', inplace=True).

Creating additional features

```
data['Previous_Close'] = data['Close'].shift(1)
data['High_Low_Diff'] = data['High'] - data['Low']
data['Open_Close_Diff'] = data['Open'] - data['Close']
```

- o Previous_Close: Shifts the 'Close' price down by 1 day to represent the previous day's closing price.
- o High_Low_Diff: Calculates the difference between 'High' and 'Low' prices.
- o Open_Close_Diff: Calculates the difference between 'Open' and 'Close' prices.

Drop rows with NaN values

```
data.dropna(inplace=True)
```

Drops rows with NaN values from the DataFrame.

Define features (X) and target (y)

```
X = data[['Open', 'High', 'Low', 'Volume', 'Previous_Close', 'High_Low_Diff', 'Open_Close_Di
y = data['Close']
```

Sets X as a DataFrame containing the predictor variables/features: 'Open', 'High', 'Low', 'Volume', 'Previous_Close', 'High_Low_Diff', and 'Open_Close_Diff'.

- Sets y as a Series containing the target variable: 'Close' price.

Split the data into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Splits the data into training (X_train, y_train) and testing (X_test, y_test) sets using train_test_split from sklearn.model_selection.
- Uses 80% of the data for training and 20% for testing (test_size=0.2), with a random seed (random_state=42) for reproducibility.

Scale the features

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Initializes a StandardScaler object to standardize (mean = 0 and variance = 1) the features.

- Applies fit_transform to the training data (X_train) and transform to the test data (X_test).

Step 3: Model Training

```
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train_scaled, y_train)
```

- Initializes a Random Forest Regressor model with 100 estimators (n_estimators=100) and a random seed (random_state=42).
- Fits the model to the scaled training data (X_train_scaled, y_train) using fit.

Step 4: Model Evaluation

```
y_pred = model.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'Mean Absolute Error: {mae}')
```

Predicts the target variable (y_pred) using the trained model on the scaled test data (X_test_scaled).

- Computes Mean Squared Error (MSE) and Mean Absolute Error (MAE) between y_test (actual values) and y_pred.
- Prints out the MSE and MAE.

Step 5: Make Predictions on Entire Dataset

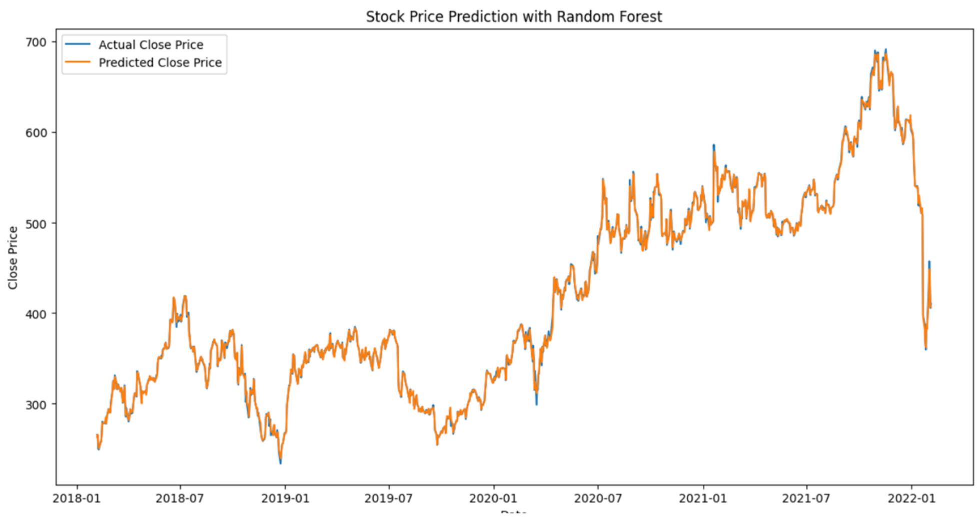
```
X_scaled = scaler.transform(X)
data['Predicted_Close'] = model.predict(X_scaled)
```

- Applies the same scaling transformation (transform) to the entire feature set (X) to make predictions on the entire dataset (X_scaled).
- Adds a new column 'Predicted_Close' to the DataFrame (data) containing the predicted closing prices.

Plot the actual vs predicted prices

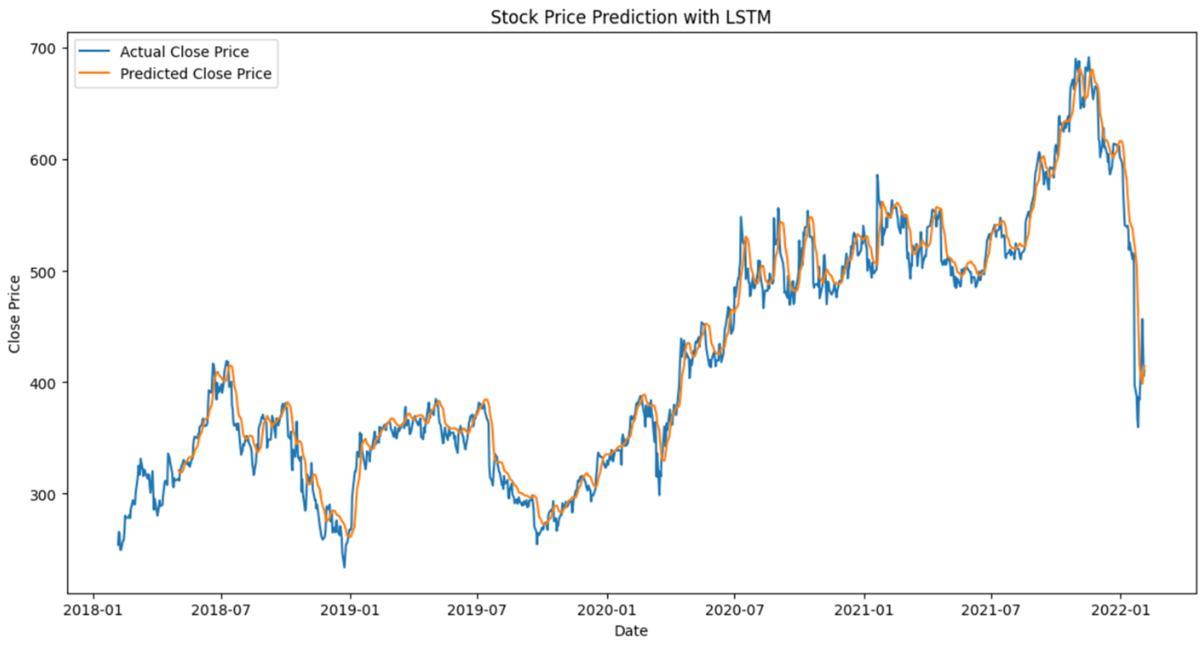
```
plt.figure(figsize=(14, 7))
plt.plot(data.index, data['Close'], label='Actual Close Price')
plt.plot(data.index, data['Predicted_Close'], label='Predicted Close Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Stock Price Prediction with Random Forest')
plt.legend()
plt.show()
```

- Creates a plot using matplotlib.pyplot to visualize the actual ('Close') and predicted ('Predicted_Close') closing prices over time.
- Labels the x-axis as 'Date' and the y-axis as 'Close Price'.
- Sets the title of the plot as 'Stock Price Prediction with Random Forest' and displays a legend.



LSTM Model:

- The code preprocesses the data and scales the 'Close' prices.
- Sequences of 60 days are created to use as input for the LSTM.
- The data is split into training and testing sets.
- An LSTM model is built and trained on the sequences.
- The model is evaluated and predictions are plotted.



CONCLUSION:

In conclusion, the use of Random Forest Regression and LSTM models for stock price prediction outperforms Linear Regression. Random Forest specializes at capturing nonlinear relationships and interactions, whereas LSTM models temporal dependencies in time series data. These advanced machine learning approaches boost forecasting accuracy significantly, emphasizing their importance in making educated decisions in dynamic financial markets. Further adjustments to model tuning and data integration could improve their predictive powers, providing significant information for investors and stakeholders looking for reliable financial forecasting and risk management tools.