**Develop neural network-based time series forecasting model.**

**EX:No.9**

**DATE:12/04/25**

**AIM:**

To develop a neural network-based time series forecasting model using LSTM on AAPL stock data.

**ALGORITHM:**

 Import AAPL stock data and necessary libraries (Pandas, Keras, etc.).

 Preprocess the data: sort by date, normalize, and create sequences.

 Split the dataset into training and testing sets.

 Reshape the data to fit LSTM input format: [samples, time\_steps, features].

 Build the LSTM model with layers and compile it.

 Train the model using the training set with validation split.

 Predict on test data, inverse-transform predictions, and evaluate performance.

**CODE:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.callbacks import EarlyStopping

# 1. Load the dataset

data = pd.read\_csv('/content/AAPL.csv')

data['Date'] = pd.to\_datetime(data['Date'])

data = data.sort\_values('Date')

data.set\_index('Date', inplace=True)

# 2. Preprocess - use only 'Close' prices

close\_data = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler()

scaled\_data = scaler.fit\_transform(close\_data)

# 3. Create sequences for LSTM

def create\_sequences(data, window\_size):

X, y = [], []

for i in range(len(data) - window\_size):

X.append(data[i:i + window\_size])

y.append(data[i + window\_size])

return np.array(X), np.array(y)

window\_size = 60

X, y = create\_sequences(scaled\_data, window\_size)

# 4. Split into train and test sets

split = int(len(X) \* 0.8)

X\_train, X\_test = X[:split], X[split:]

y\_train, y\_test = y[:split], y[split:]

# 5. Build the LSTM model

model = Sequential()

model.add(LSTM(units=64, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)))

model.add(Dropout(0.2))

model.add(LSTM(units=32))

model.add(Dropout(0.2))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mean\_squared\_error')

# 6. Train the model

early\_stop = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=16,

validation\_split=0.1, callbacks=[early\_stop], verbose=1)

# 7. Predict and inverse scale

predicted = model.predict(X\_test)

predicted\_prices = scaler.inverse\_transform(predicted)

actual\_prices = scaler.inverse\_transform(y\_test)

# 8. Evaluation Metrics

mae = mean\_absolute\_error(actual\_prices, predicted\_prices)

mse = mean\_squared\_error(actual\_prices, predicted\_prices)

rmse = np.sqrt(mse)

print(f"\n📈 LSTM Forecasting Metrics:")

print(f"MAE: {mae:.4f}")

print(f"MSE: {mse:.4f}")

print(f"RMSE: {rmse:.4f}")

# 9. Plot predictions vs actual

plt.figure(figsize=(12, 6))

plt.plot(actual\_prices, label='Actual Prices')

plt.plot(predicted\_prices, label='Predicted Prices')

plt.title('AAPL Stock Price Prediction using LSTM')

plt.xlabel('Days')

plt.ylabel('Price')

plt.legend()

plt.show()

# 10.Plot training history

plt.figure(figsize=(10, 4))

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Model Loss Over Epochs')

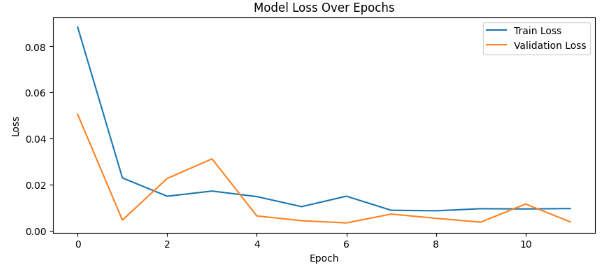
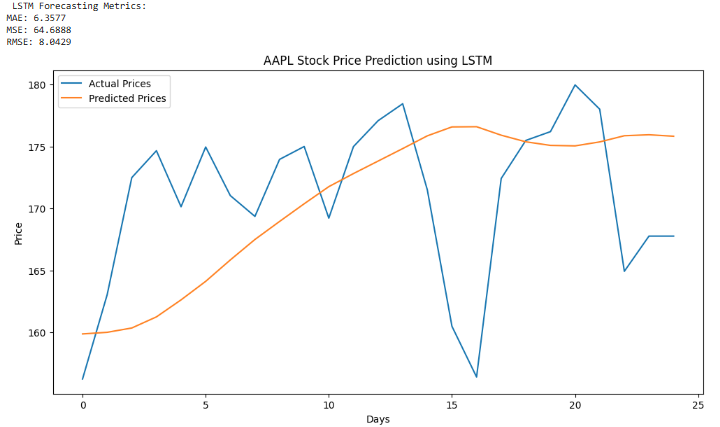
plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.show()

**OUTPUT:**

****

**RESULT:**

Thus the program has been completed and verified successfully.