**Develop neural network-based time series forecasting model run in python code**

**EX.No:9**

**DATE:**

**AIM:**

To build an LSTM-based model for forecasting future values in a time series using historical data.

**ALGORITHM:**

1. Load and preprocess the time series data.
2. Normalize the data and create input sequences.
3. Split the data into training and testing sets.
4. Build and train the LSTM model.
5. Make predictions and evaluate performance.

**CODE:**

# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.metrics import mean\_squared\_error

# Step 1: Load the dataset

file\_path = 'Dataset.csv'

data = pd.read\_csv(file\_path)

# Step 2: Prepare the date column

data['month'] = data['month'] + '-2020'  # Add dummy year

data['month'] = pd.to\_datetime(data['month'], format='%d-%b-%Y')

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

# Step 3: Handle missing values

data['price'] = data['price'].interpolate()

# Step 4: Normalize the data

scaler = MinMaxScaler()

data['price\_scaled'] = scaler.fit\_transform(data[['price']])

# Step 5: Create lag features

def create\_dataset(series, n\_lags=3):

    X, y = [], []

    for i in range(n\_lags, len(series)):

        X.append(series[i-n\_lags:i])

        y.append(series[i])

    return np.array(X), np.array(y)

n\_lags = 3

X, y = create\_dataset(data['price\_scaled'].values, n\_lags)

# Step 6: Train-Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)

# Step 7: Build the Neural Network

model = Sequential()

model.add(Dense(64, activation='relu', input\_dim=n\_lags))

model.add(Dense(32, activation='relu'))

model.add(Dense(1))  # Output layer

model.compile(optimizer='adam', loss='mse')

# Step 8: Train the model

history = model.fit(X\_train, y\_train, epochs=100, validation\_data=(X\_test, y\_test), verbose=0)

# Step 9: Predict

y\_pred = model.predict(X\_test)

# Inverse transform to original scale

y\_test\_inv = scaler.inverse\_transform(y\_test.reshape(-1, 1))

y\_pred\_inv = scaler.inverse\_transform(y\_pred)

# Step 10: Plot Actual vs Predicted

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

plt.plot(y\_test\_inv, label='Actual')

plt.plot(y\_pred\_inv, label='Predicted', color='red')

plt.title('Actual vs Predicted')

plt.xlabel('Time')

plt.ylabel('Price')

plt.legend()

plt.grid(True)

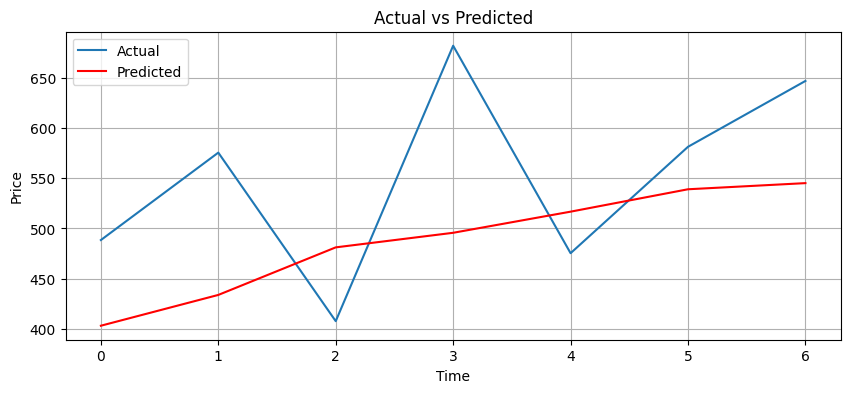
plt.show()

# Step 11: Evaluate

rmse = np.sqrt(mean\_squared\_error(y\_test\_inv, y\_pred\_inv))

print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')

**OUTPUT:**



**\**

**RESULT:**

Thus the program has been completed and verified successfully.