**Exp no: 9 Develop neural network-based time series forecasting model.**

**Date: 10/4/25**

**Aim:**

The aim of this project is to develop a neural network-based time series forecasting model using Long Short-Term Memory (LSTM) networks to accurately predict future air passenger traffic based on historical data from the AirPassengers dataset.

**Objectives:**

The primary objective is to preprocess and transform the time series data, train an LSTM model to capture temporal dependencies, and evaluate its performance in forecasting future passenger counts. The project also aims to visualize predicted versus actual values and demonstrate the effectiveness of LSTM in modeling sequential data with seasonality and trends.

**Background/Scope:**

Forecasting time series data, especially in domains like air travel, is critical for planning and decision-making. Traditional methods like ARIMA often struggle with complex patterns and long-term dependencies. This project explores the scope of deep learning, particularly LSTM networks, which are well-suited for handling sequential data. Using the AirPassengers dataset, which records monthly totals of international airline passengers from 1949 to 1960, this project demonstrates how neural networks can be leveraged for accurate and scalable forecasting in real-world scenarios.

**Steps for Time Series Sales Data Preprocessing:**

**Step 1: Load and Prepare the Data**

Import required libraries, load the dataset, convert the 'Month' column to datetime format, and set it as the index to structure it for time series analysis. This step ensures that the data is properly formatted for temporal operations and modeling.

import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.tsa.arima.model import ARIMA

from pandas.plotting import autocorrelation\_plot

from statsmodels.tsa.stattools import adfuller

path = '/content/AirPassengers.csv'

df = pd.read\_csv(path)

df['Month'] = pd.to\_datetime(df['Month'])

df.set\_index('Month', inplace=True)

**Step 2: Normalize the Data**

We normalize the data using **MinMaxScaler** to scale the values between 0 and 1. This is essential for neural networks to perform optimally.

scaler = MinMaxScaler()

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

**Step 3: Prepare Sequences for LSTM Input**

LSTM models require sequences of data as input. Here, we create sequences from the dataset by using a sliding window approach, where the previous time\_step number of observations are used to predict the next one.

def create\_sequences(data, time\_step=10):

X, y = [], []

for i in range(len(data) - time\_step):

X.append(data[i:i + time\_step])

y.append(data[i + time\_step])

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

time\_step = 10

X, y = create\_sequences(scaled\_data, time\_step)

**Step 4: Split the Data into Training and Testing Sets**

We split the data into **training** and **testing** sets, typically using 80% for training and 20% for testing.

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

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

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

**Step 5: Build and Train the LSTM Model**

We define the architecture of the LSTM model. The model is compiled with the Adam optimizer and mean squared error loss function, and it is then trained on the training data.

model = Sequential()

model.add(LSTM(50, return\_sequences=False, input\_shape=(X\_train.shape[1], 1)))

model.add(Dense(1))

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

# Train the model

model.fit(X\_train, y\_train, epochs=100, batch\_size=16, verbose=1)

**Step 6: Make Predictions and Visualize Results**

Once the model is trained, we use it to make predictions on both the training and testing sets. Afterward, we visualize the predicted and actual values to evaluate the model’s performance.

train\_pred = model.predict(X\_train)

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

# Inverse transform to get actual values

train\_pred = scaler.inverse\_transform(train\_pred)

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

test\_pred = scaler.inverse\_transform(test\_pred)

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

# Plot the results

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

plt.plot(df.index[time\_step:split + time\_step], y\_train\_inv, label='Train Actual')

plt.plot(df.index[time\_step:split + time\_step], train\_pred, label='Train Predicted')

plt.plot(df.index[split + time\_step:], y\_test\_inv, label='Test Actual')

plt.plot(df.index[split + time\_step:], test\_pred, label='Test Predicted')

plt.legend()

plt.title('AirPassengers Forecast using LSTM')

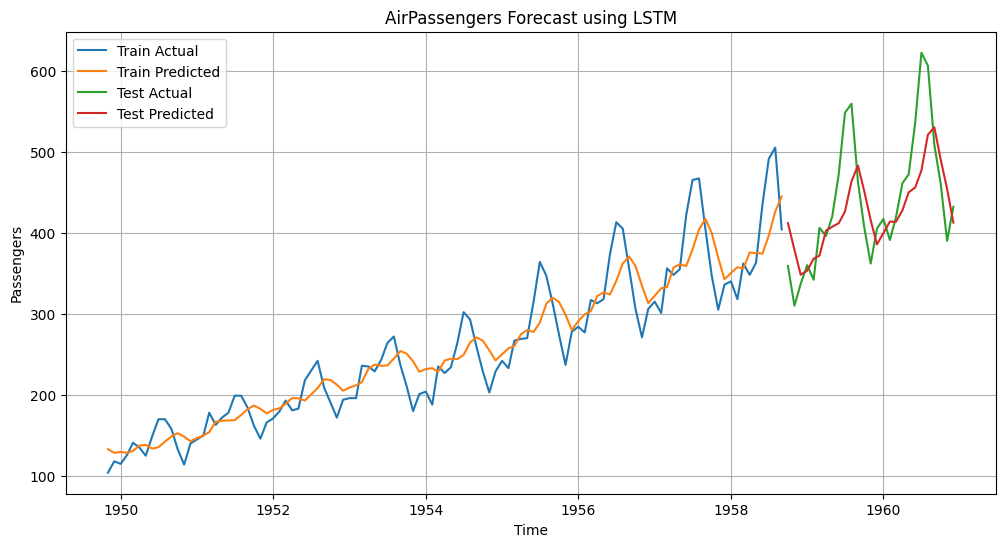
plt.xlabel('Time')

plt.ylabel('Passengers')

plt.grid(True)

plt.show()

**Output:**



**Result:**

Neural network-based time series forecasting model for Air passenger dataset has been developed successfully.