**Assignment – 1**

import pandas as pd

df = pd.read\_csv(r"C:\Users\dell\Downloads\boston\_housing.csv")

print(df.head())

df.isnull().sum()

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

X = df.loc[:, df.columns != 'MEDV']

y = df.loc[:, df.columns == 'MEDV']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)

from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()

mms.fit(X\_train)

X\_train = mms.transform(X\_train)

X\_test = mms.transform(X\_test)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Dense(128, input\_shape=(13, ), activation='relu', name='dense\_1'))

model.add(Dense(64, activation='relu', name='dense\_2'))

model.add(Dense(1, activation='linear', name='dense\_output'))

model.compile(optimizer='adam', loss='mse', metrics=['mae'])

model.summary()

history = model.fit(X\_train, y\_train, epochs=100, validation\_split=0.05, verbose = 1)

mse, mae = model.evaluate(X\_test, y\_test)

print('MSE: ', mse)

print('MAE: ', mae)

y1 = model.predict(X\_test[:])

y\_test

ps=[]

for i in y1:

ps.append(list(i)[0])

d = pd.DataFrame({'actual':y\_test['MEDV'],'predicted':ps})

d

from sklearn.metrics import r2\_score

r2 = r2\_score(y\_test, y1)

print('R² Score:', r2)

**Assignment\_no =02**

from tensorflow.keras.datasets import imdb

(train\_data, train\_label), (test\_data, test\_label) = imdb.load\_data(num\_words = 10000)

import numpy as np

def vectorize\_sequences(sequences, dimensions = 10000):

results = np.zeros((len(sequences), dimensions))

for i,sequences in enumerate(sequences):

results[i, sequences] = 1

return results

x\_train = vectorize\_sequences(train\_data)

x\_test = vectorize\_sequences(test\_data)

y\_train = np.asarray(train\_label).astype('float32')

y\_test = np.asarray(test\_label).astype('float32')

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Dense(16, input\_shape=(10000, ), activation = "relu"))

model.add(Dense(16, activation = "relu"))

model.add(Dense(1, activation = "sigmoid"))

model.compile(optimizer='adam', loss = 'mse', metrics = ['accuracy'])

model.summary()

history = model.fit(x\_train, y\_train, validation\_split = 0.2, epochs = 20, verbose = 1, batch\_size = 512)

mse,mae = model.evaluate(x\_test,y\_test)

print('MSE ',mse)

print('MAE ',mae)

y\_preds = model.predict(x\_test)

preds=[]

for i in y\_preds:

if i[0]>0.5:

preds.append(1)

else:

preds.append(0)

from sklearn.metrics import accuracy\_score,precision\_score,recall\_score

print(accuracy\_score(y\_test,preds))

print(precision\_score(y\_test,preds))

print(recall\_score(y\_test,preds))

word\_index = imdb.get\_word\_index()

def return\_token(tid):

for k,v in word\_index.items():

if v == tid-3:

return k

return '?'

def print\_review(id\_):

sentence = ' '.join(return\_token(i) for i in train\_data[id\_])

return sentence

print\_review(0)

train\_label[0]

print\_review(1)

train\_label[1]

**Assignment\_no=3**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

df = pd.read\_csv(r"C:\Users\dell\Downloads\GOOG.csv")

df.isnull().sum()

df = df[['date', 'close']]

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

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

df

# Scale the data

scaler = MinMaxScaler()

scaled\_data = scaler.fit\_transform(df[['close']])

# Visualize the closing price

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

plt.plot(df['close'], label='Google Stock Price')

plt.title('Google Stock Price Over Time')

plt.xlabel('Date')

plt.ylabel('Close Price USD')

plt.legend()

plt.show()

# Create sequences

def create\_sequences(data, seq\_length):

X, y = [], []

for i in range(len(data) - seq\_length):

X.append(data[i:i + seq\_length])

y.append(data[i + seq\_length])

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

sequence\_length = 60

X, y = create\_sequences(scaled\_data, sequence\_length)

# Train-test split

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

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

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

# Build the LSTM model

model = Sequential([

LSTM(50, return\_sequences=False, input\_shape=(sequence\_length, 1)),

Dense(1)

])

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

model.summary()

# Train the model

# Train the model

history = model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_data=(X\_test, y\_test))

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

# Inverse scale predictions

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

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

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

# Evaluation

mae = mean\_absolute\_error(y\_test\_rescaled, y\_pred\_rescaled)

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

r2 = r2\_score(y\_test\_rescaled, y\_pred\_rescaled)

mape = np.mean(np.abs((y\_test\_rescaled - y\_pred\_rescaled) / y\_test\_rescaled)) \* 100

print(f"\n📈 Evaluation Metrics:")

print(f"Mean Absolute Error (MAE): {mae:.2f}")

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

print(f"R² Score: {r2:.4f}")

print(f"Mean Absolute Percentage Error (MAPE): {mape:.2f}%")

# Plot actual vs predicted

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

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

plt.plot(y\_pred\_rescaled, label='Predicted Price')

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

plt.xlabel('Time')

plt.ylabel('Price USD')

plt.legend()

plt.show()