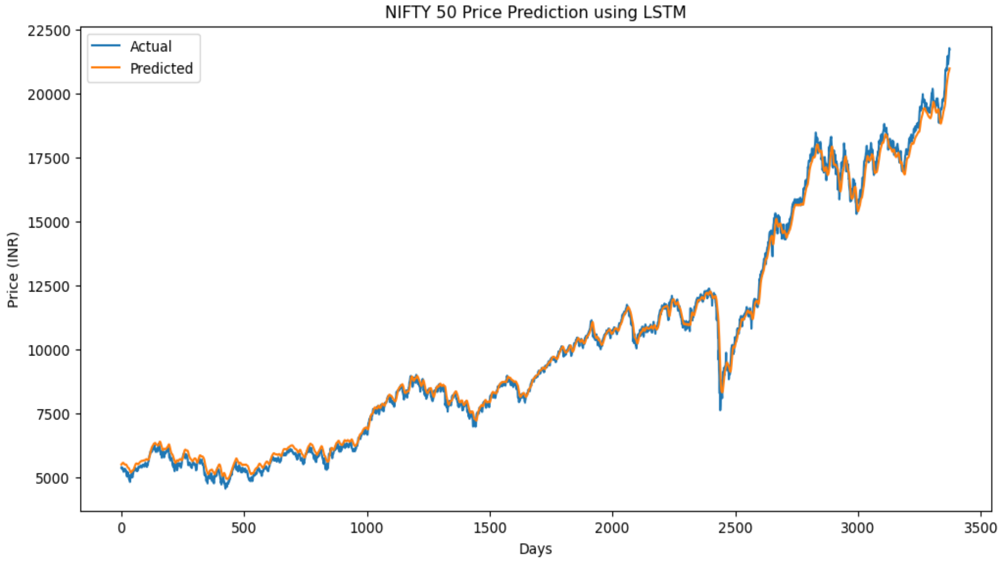
PROGRAM

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
df = yf.download("^NSEI", start="2010-01-01", end="2024-01-01")
df = df[['Close']]
df.dropna(inplace=True)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df)
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(seq_length, len(data)):
        X.append(data[i - seq_length:i, 0])
        y.append(data[i, 0])
    return np.array(X), np.array(y)
seq_length = 60  # 60 days look-back
X, y = create_sequences(scaled_data, seq_length)
X = np.reshape(X, (X.shape[0], X.shape[1], 1))
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()

history = model.fit(X, y, epochs=20, batch_size=32)

last_sequence = scaled_data[-seq_length:]

last_sequence = np.reshape(last_sequence, (1, seq_length, 1))

predicted_price = model.predict(last_sequence)

predicted_price = scaler.inverse_transform(predicted_price)

print(f"Predicted Next Day Closing Price: ₹{predicted_price[0][0]:.2f}")

train_predict = model.predict(X)

train_predict = scaler.inverse_transform(train_predict)

y_actual = scaler.inverse_transform(y.reshape(-1, 1))

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

plt.plot(y_actual, label='Actual')

plt.plot(train_predict, label='Predicted')

plt.title('NIFTY 50 Price Prediction using LSTM')

plt.xlabel('Days')

plt.ylabel('Price (INR)')

plt.legend()

plt.show()
```

<u>OUTPUT</u>



PROGRAM

```
!pip install transformers torch

from transformers import pipeline

def analyze_sentiment(text):

   sentiment_pipeline = pipeline("sentiment-analysis", model="distilbert-base-uncased-finetuned-sst-2-english")

   result = sentiment_pipeline(text)[0]

   return result

# Example 1: Positive sentiment

positive_sentence = "I absolutely love this new phone, it's incredible!"

positive_result = analyze_sentiment(positive_sentence)

print(f"Sentence: '{positive_sentence}'")

print(f"Sentiment: {positive_result['label']} (Score: {positive_result['score']:.4f})")

print("-" * 20)

#Example 2: Negative sentiment

negative_sentence = "This movie was a complete waste of time, I'm so disappointed."

negative_result = analyze_sentiment(negative_sentence)

print(f"Sentence: '{negative_sentence}'")

print(f"Sentiment: {negative_result['label']} (Score: {negative_result['score']:.4f})")

print("-" * 20)

# Example 3: Neutral/Mixed sentiment

neutral_sentence = "The food was okay, but the service was terrible."

neutral_result = analyze_sentiment(neutral_sentence)

print(f"Sentence: '{neutral_sentence}'")

print(f"Sentiment: {neutral_result['label']} (Score: {neutral_result['score']:.4f})")
```

OUTPUT

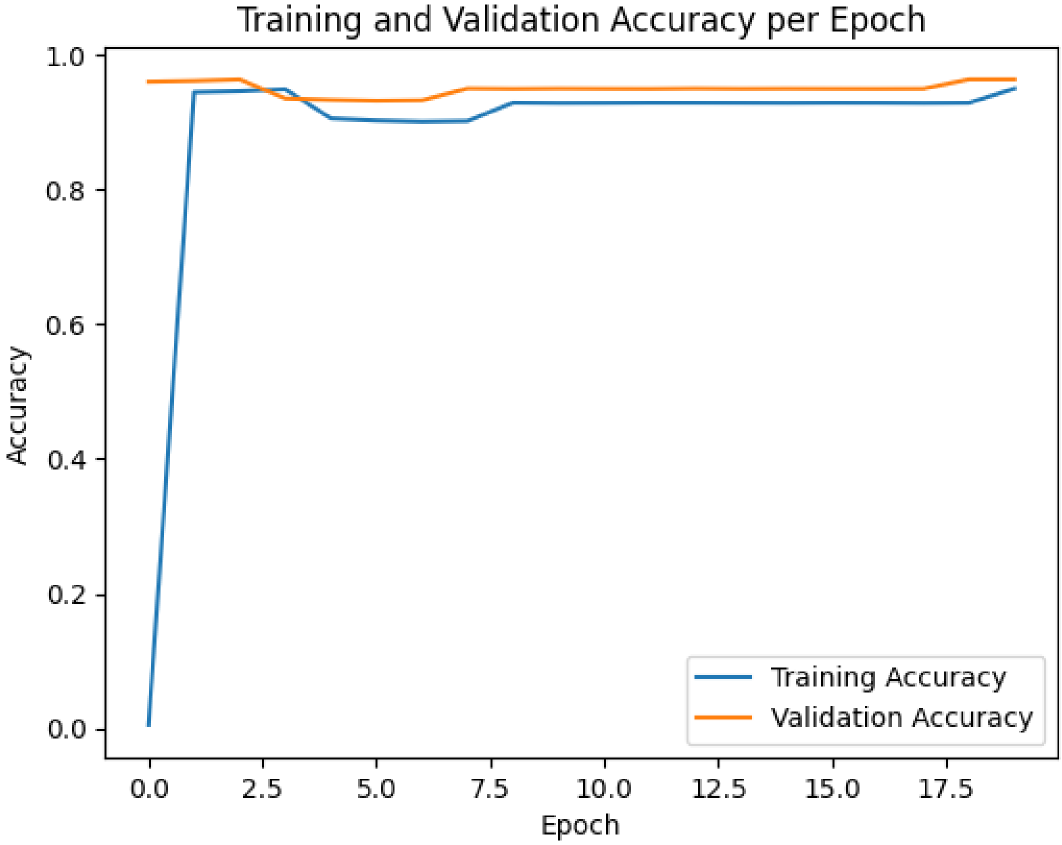
```
Sentence: 'I absolutely love this new phone, it's incredible!'

Sentiment: POSITIVE (Score: 0.9999)


Sentence: 'This movie was a complete waste of time, I'm so disappointed.'

Sentiment: NEGATIVE (Score: 0.9998)


Sentence: 'The food was okay, but the service was terrible.'

Sentiment: NEGATIVE (Score: 0.9862)
```

PROGRAM

```
import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, LSTM, Dense, Embedding

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences

from sklearn.model_selection import train_test_split

from tensorflow.keras.utils import to_categorical

import matplotlib.pyplot as plt

file_path ='/content/hindi_english_parallel.csv'

english_sentences, hindi_sentences = [], []

with open(file_path, 'r', encoding='latin-1') as f:

    for line in f:

        parts = line.split("\t")

        if len(parts) == 3:

            eng, hin, _ = parts

            english_sentences.append(eng.strip())

            hindi_sentences.append(hin.strip())

english_tokenizer = Tokenizer(char_level=True)

hindi_tokenizer = Tokenizer(char_level=True)

english_tokenizer.fit_on_texts(english_sentences)

hindi_tokenizer.fit_on_texts(hindi_sentences)

eng_sequences = english_tokenizer.texts_to_sequences(english_sentences)

hin_sequences = hindi_tokenizer.texts_to_sequences(hindi_sentences)

max_eng_len = max(len(seq) for seq in eng_sequences)

max_hin_len = max(len(seq) for seq in hin_sequences)

eng_padded = pad_sequences(eng_sequences, maxlen=max_eng_len, padding='post')

hin_padded = pad_sequences(hin_sequences, maxlen=max_hin_len, padding='post')

num_decoder_tokens = len(hindi_tokenizer.word_index) + 1

hin_padded_shifted = np.array([to_categorical(seq, num_classes=num_decoder_tokens) for seq in
hin_padded[:, 1:]])
```

```
eng_train, eng_test, hin_train, hin_test = train_test_split(eng_padded, hin_padded,
test_size=0.2,random_state=42)
hin_train_shifted, hin_test_shifted = train_test_split(hin_padded_shifted,
test_size=0.2,random_state=42)
latent_dim=256
encoder_inputs = Input(shape=(max_eng_len,))
encoder_embedding = Embedding(input_dim=len(english_tokenizer.word_index) + 1,
output_dim=latent_dim)(encoder_inputs)
encoder_outputs, state_h, state_c = LSTM(latent_dim, return_state=True)(encoder_embedding)
encoder_states = [state_h, state_c]
decoder_inputs = Input(shape=(max_hin_len - 1,))
decoder_embedding = Embedding(input_dim=num_decoder_tokens,
output_dim=latent_dim)(decoder_inputs)
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_embedding, initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=["accuracy"])
batch_size = 64
epochs = 20
history = model.fit([eng_train, hin_train[:, :-1]], hin_train_shifted,batch_size=batch_size,
epochs=epochs, validation_split=0.2)
model.save("eng_to_hin_translation_model.h5")
loss, accuracy = model.evaluate([eng_test, hin_test[:, :-1]], hin_test_shifted)
print(f"Test Accuracy: {accuracy * 100:.2}%")


plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title("Training and Validation Accuracy per Epoch")
plt.show()
```

OUTPUT


Training and Validation Accuracy per Epoch
1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
Training Accuracy
Validation Accuracy