DEEPLEARNING PROJECT

API KEY : B9I2SS3DP9ZKFC9Y

TIME_SERIES_INTRADAY

```
import requests

# replace the "demo" apikey below with your own key from https://www.alphavantage.co/support/#api-key
url = 'https://www.alphavantage.co/query?function=TIME_SERIES_INTRADAY&symbol=IBM&interval=5min&apikey=demo'
r = requests.get(url)
data = r.json()

print(data)
```

TIME_SERIES_DAILY

```
import requests

# replace the "demo" apikey below with your own key from https://www.alphavantage.co/support/#api-key
url = 'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=IBM&apikey=demo'
r = requests.get(url)
data = r.json()

print(data)
```

```
import requests
import pandas as pd
from flask import Flask, render_template, request

# Define your Alpha Vantage API key
api_key = 'B9I2SS3DP9ZKFC9Y'

# Initialize Flask
app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        # Get the stock symbol entered by the user
        symbol = request.form['symbol'].strip().upper()

        # Define the API endpoint URL with the user-provided symbol
        api_url = f'https://www.alphavantage.co/query?function=TIME_SERIES_INTRADAY' \
                  f'&symbol={symbol}&interval=1min&apikey={api_key}'

        try:
            # Make a GET request to the Alpha Vantage API
            response = requests.get(api_url)

            # Check if the request was successful (status code 200)
            if response.status_code == 200:
                # Parse the JSON response data
                data = response.json()

                # Check if the 'Time Series (1min)' key exists in the response data
                if 'Time Series (1min)' in data:
                    # Extract the intraday stock price data
                    df = pd.DataFrame.from_dict(data['Time Series (1min)'], orient='index')
                    df.index = pd.to_datetime(df.index)
                    df['4. close'] = df['4. close'].astype(float)

                    # Calculate the Simple Moving Average (SMA) with a 10-minute window
                    sma_window = 10
```

```
                    df['SMA'] = df['4. close'].rolling(window=sma_window).mean()

                    # Get the last calculated SMA value
                    last_sma = df['SMA'].iloc[-1]

                    return render_template('index.html', symbol=symbol, last_sma=last_sma)
                else:
                    return render_template('index.html', error=f"No data found for {symbol}.")
            else:
                return render_template('index.html', error=f"API request failed with status code {response.status_code}")
        except requests.exceptions.RequestException as e:
            return render_template('index.html', error=f"Error: {e}")
    return render_template('index.html')

if __name__ == '__main__':
    app.run(debug=True)
```

```
import numpy as np
import pandas as pd
from alpha_vantage.timeseries import TimeSeries
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
import matplotlib.pyplot as plt

# Define your Alpha Vantage API key
api_key = 'B9I2SS3DP9ZKFC9Y'

# Define the stock symbol and time period
symbol = 'AAPL'
interval = '1d'  # Daily data

# Initialize Alpha Vantage API client
ts = TimeSeries(key=api_key, output_format='pandas')

# Fetch historical stock price data
data, meta_data = ts.get_daily(symbol=symbol, outputsize='full')
data = data.rename(columns={"1. open": "Open", "2. high": "High", "3. low": "Low", "4. close": "Close", "5. volume": "Volume"})
data = data.sort_index(ascending=True)

# Extract the 'Close' prices for prediction
prices = data['Close'].values
prices = prices.reshape(-1, 1)

# Normalize the data using Min-Max scaling
scaler = MinMaxScaler(feature_range=(0, 1))
prices_normalized = scaler.fit_transform(prices)

# Split the data into training and testing sets
train_size = int(len(prices_normalized) * 0.8)
train_data = prices_normalized[:train_size]
test_data = prices_normalized[train_size:]

# Create sequences for LSTM training
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)

seq_length = 10  # Adjust as needed
X_train, y_train = create_sequences(train_data, seq_length)
X_test, y_test = create_sequences(test_data, seq_length)

# Build an LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(seq_length, 1)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

```
model.summary()

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)

# Make predictions
train_predictions = model.predict(X_train)
test_predictions = model.predict(X_test)

# Inverse transform the predictions to original scale
train_predictions = scaler.inverse_transform(train_predictions)
test_predictions = scaler.inverse_transform(test_predictions)

# Calculate RMSE (Root Mean Squared Error) on the test set
test_rmse = np.sqrt(mean_squared_error(prices[-len(test_predictions):], test_predictions))
print(f'Test RMSE: {test_rmse}')

# Visualize the predictions
plt.figure(figsize=(12, 6))
plt.plot(prices[-len(test_predictions):], label='True Prices')
plt.plot(test_predictions, label='Predicted Prices')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.title(f'{symbol} Stock Price Prediction (Test Data)')
plt.show()
```

symbol = input("Enter the stock symbol (e.g., AAPL): ").strip().upper()