import streamlit as st import yfinance as yf import pandas as pd import numpy as np from datetime import datetime import matplotlib.pyplot as plt from sklearn.ensemble import RandomForestRegressor from sklearn.linear\_model import LinearRegression from sklearn.model\_selection import train\_test\_split from sklearn.metrics import mean\_squared\_error, r2\_score from tensorflow.keras.models import Sequential from tensorflow.keras.layers import LSTM, Dense from sklearn.preprocessing import MinMaxScaler import warnings

warnings.filterwarnings("ignore")

st.title(" Stock Price Prediction and Recommendation System")

# Sidebar Inputs st.sidebar.header("User Input") company = st.sidebar.text\_input("Enter stock ticker (e.g., AAPL, MSFT):", "AAPL").upper() start\_date = st.sidebar.date\_input("Start date", datetime(2010, 1, 1)) end\_date = st.sidebar.date\_input("End date", datetime.today()) run\_lstm = st.sidebar.checkbox("Include LSTM (slower)", value=False)

@st.cache\_data def load\_data(ticker, start, end):     data = yf.download(ticker, start=start, end=end, progress=False)     data.reset\_index(inplace=True)     return data

if company:     st.subheader(f"Fetching data for {company}...")     data = load\_data(company, start\_date, end\_date)

    if data.empty:         st.error(" No data found. Please check the stock ticker or date range.")         st.stop()

    # Preprocessing     df = data[['Date', 'Close']].copy()     df['Date'] = pd.to\_datetime(df['Date'])     df['MA7'] = df['Close'].rolling(window=7).mean()     df['MA20'] = df['Close'].rolling(window=20).mean()

    df['MA50'] = df['Close'].rolling(window=50).mean()     df['Target'] = df['Close'].shift(-1)     df.dropna(inplace=True)

    features = ['Close', 'MA7', 'MA20', 'MA50']     X = df[features]     y = df['Target']

    # Train/Test Split     X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    with st.spinner("Training models..."):         # Random Forest         rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)         rf\_model.fit(X\_train, y\_train)         rf\_pred = rf\_model.predict(X\_test)         rf\_rmse = np.sqrt(mean\_squared\_error(y\_test, rf\_pred))         rf\_r2 = r2\_score(y\_test, rf\_pred)

        # Linear Regression         lr\_model = LinearRegression()         lr\_model.fit(X\_train, y\_train)         lr\_pred = lr\_model.predict(X\_test)         lr\_rmse = np.sqrt(mean\_squared\_error(y\_test, lr\_pred))         lr\_r2 = r2\_score(y\_test, lr\_pred)

        # Optional: LSTM         if run\_lstm:             scaler = MinMaxScaler()             scaled\_data = scaler.fit\_transform(df[features + ['Target']])             seq\_length = 10             X\_lstm, y\_lstm = [], []             for i in range(seq\_length, len(scaled\_data)):                 X\_lstm.append(scaled\_data[i-seq\_length:i, :-1])                 y\_lstm.append(scaled\_data[i, -1])             X\_lstm, y\_lstm = np.array(X\_lstm), np.array(y\_lstm)             split = int(0.8 \* len(X\_lstm))             X\_lstm\_train, y\_lstm\_train = X\_lstm[:split], y\_lstm[:split]             X\_lstm\_test, y\_lstm\_test = X\_lstm[split:], y\_lstm[split:]

            lstm\_model = Sequential([                 LSTM(50, return\_sequences=True, input\_shape=(X\_lstm\_train.shape[1], X\_lstm\_train.shape[2])),                 LSTM(50),                 Dense(1)             ])

            lstm\_model.compile(optimizer='adam', loss='mean\_squared\_error')             lstm\_model.fit(X\_lstm\_train, y\_lstm\_train, epochs=10, batch\_size=32, verbose=0)

            lstm\_pred\_scaled = lstm\_model.predict(X\_lstm\_test, verbose=0)             # Proper inverse transform using padding for unscaled features             dummy = np.zeros((lstm\_pred\_scaled.shape[0], scaled\_data.shape[1]-1))             lstm\_pred = scaler.inverse\_transform(np.hstack((dummy, lstm\_pred\_scaled)))[:, -1]             y\_lstm\_true = scaler.inverse\_transform(np.hstack((dummy, y\_lstm\_test.reshape(-1, 1))))[:, -1]             lstm\_rmse = np.sqrt(mean\_squared\_error(y\_lstm\_true, lstm\_pred))             lstm\_r2 = r2\_score(y\_lstm\_true, lstm\_pred)

    # Display Metrics     st.subheader(" Model Performance")

    col1, col2, col3 = st.columns(3)     col1.metric("RF RMSE", f"{rf\_rmse:.2f}", f"R²: {rf\_r2:.3f}")     col2.metric("LR RMSE", f"{lr\_rmse:.2f}", f"R²: {lr\_r2:.3f}")     if run\_lstm:         col3.metric("LSTM RMSE", f"{lstm\_rmse:.2f}", f"R²: {lstm\_r2:.3f}")

    # Comparison Bar Chart     st.subheader(" RMSE and R² Comparison")     fig, ax = plt.subplots(1, 2, figsize=(14, 5))     models = ['Random Forest', 'Linear Regression']     rmse\_values = [rf\_rmse, lr\_rmse]     r2\_values = [rf\_r2, lr\_r2]     if run\_lstm:         models.append('LSTM')         rmse\_values.append(lstm\_rmse)         r2\_values.append(lstm\_r2)

    ax[0].bar(models, rmse\_values, color=['blue', 'green', 'orange'][:len(models)])     ax[0].set\_title("RMSE")     ax[1].bar(models, r2\_values, color=['blue', 'green', 'orange'][:len(models)])     ax[1].set\_title("R² Score")     st.pyplot(fig)

    # Plot RF Prediction     st.subheader(f"{company} - Last 60 Days Prediction (Random Forest)")     last\_n = 60     plot\_dates = df['Date'].iloc[-len(y\_test):].values[-last\_n:]     actual\_prices = y\_test.values[-last\_n:]     predicted\_prices = rf\_pred[-last\_n:]

    fig2, ax2 = plt.subplots(figsize=(14, 5))

    ax2.plot(plot\_dates, actual\_prices, label='Actual', color='blue', marker='o')     ax2.plot(plot\_dates, predicted\_prices, label='RF Predicted', color='orange', marker='o')     ax2.set\_xlabel("Date")     ax2.set\_ylabel("Price")     ax2.legend()     ax2.grid(True)     st.pyplot(fig2)

# ------------------------# Recommendation # ------------------------st.subheader(" Recommendation")

latest\_features = df[features].iloc[-1].values.reshape(1, -1) next\_day\_price = rf\_model.predict(latest\_features)[0] # Ensure scalar current\_price = df['Close'].iloc[-1] # Scalar

# Ensure both are floats change = float((next\_day\_price - current\_price) / current\_price \* 100)

st.write(f"Next Day Predicted Price (Random Forest): \*\*${next\_day\_price:.2f}\*\*")

if change > 5:     st.success(" Recommendation: \*\*Buy\*\* (Expected ↑ more than 5%)") elif change < -5:     st.error(" Recommendation: \*\*Sell\*\* (Expected ↓ more than 5%)") else:     st.info(" Recommendation: \*\*Hold\*\* (No major change expected)")