**Hybrid Rf Lstm Model**

import pandas as pd # For working with tabular data

import numpy as np # For numerical operations

import matplotlib.pyplot as plt # For visualizing results

import yfinance as yf # For fetching stock data from Yahoo Finance

from sklearn.preprocessing import MinMaxScaler # For scaling features

from sklearn.ensemble import RandomForestRegressor # For building Random Forest model

from sklearn.metrics import mean\_squared\_error # For evaluating model performance

from tensorflow.keras.models import Sequential # For building LSTM model

from tensorflow.keras.layers import LSTM, Dense # LSTM and Dense layers

# Download stock data using Yahoo Finance API

def load\_stock\_data(ticker, start\_date, end\_date):

data = yf.download(ticker, start=start\_date, end=end\_date) # Download stock data

return data

# Add technical indicators to enhance feature set

def add\_technical\_indicators(data):

delta = data['Close'].diff() # Calculate daily price change

gain = (delta.where(delta > 0, 0)).rolling(14).mean() # Average gain over 14 days

loss = (-delta.where(delta < 0, 0)).rolling(14).mean() # Average loss over 14 days

rs = gain / loss # Relative Strength

data['RSI'] = 100 - (100 / (1 + rs)) # Compute RSI

data['SMA\_40'] = data['Close'].rolling(window=40).mean() # 40-day SMA

data['SMA\_100'] = data['Close'].rolling(window=100).mean() # 100-day SMA

data['Upper\_Band'] = data['Close'].rolling(20).mean() + 2 \* data['Close'].rolling(20).std() # Bollinger Upper Band

data['Lower\_Band'] = data['Close'].rolling(20).mean() - 2 \* data['Close'].rolling(20).std() # Bollinger Lower Band

return data.dropna() # Remove rows with NaN values

# Scale the selected features using MinMaxScaler

def preprocess\_data(data):

scaler = MinMaxScaler() # Initialize general scaler

close\_scaler = MinMaxScaler() # Separate scaler for close price

features = ['Close', 'Volume', 'RSI', 'Upper\_Band', 'Lower\_Band', 'SMA\_40', 'SMA\_100']

data[features] = scaler.fit\_transform(data[features]) # Scale all features

data[['Close']] = close\_scaler.fit\_transform(data[['Close']]) # Scale close separately

return data, scaler, close\_scaler # Return scaled data and scalers

# Build and train a Random Forest Regressor

def train\_random\_forest(X\_train, y\_train):

rf\_model = RandomForestRegressor(n\_estimators=100) # Create model with 100 trees

rf\_model.fit(X\_train, y\_train) # Train model on training data

return rf\_model

# Build and train LSTM model

def train\_lstm(X\_train, y\_train):

X\_train\_lstm = np.reshape(X\_train.values, (X\_train.shape[0], X\_train.shape[1], 1)) # Reshape for LSTM input

lstm\_model = Sequential([

LSTM(50, return\_sequences=True, input\_shape=(X\_train\_lstm.shape[1], 1)), # First LSTM layer

LSTM(50, return\_sequences=False), # Second LSTM layer

Dense(1) # Output layer

])

lstm\_model.compile(loss='mean\_squared\_error', optimizer='adam', metrics=['mse']) # Compile model

lstm\_model.fit(X\_train\_lstm, y\_train, epochs=20, batch\_size=32, verbose=1) # Train model

return lstm\_model

# Visualize actual vs predicted stock prices

def visualize\_predictions(y\_test, final\_preds, actual\_prices):

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

plt.plot(actual\_prices.index[-len(y\_test):], actual\_prices[-len(y\_test):], label="Actual Prices", color='blue')

plt.plot(actual\_prices.index[-len(y\_test):], final\_preds, label="Predicted Prices", color='red')

plt.title("Actual Stock Prices vs Predicted Stock Prices")

plt.xlabel("Date")

plt.ylabel("Stock Price")

plt.legend()

plt.show()

# Main script to execute hybrid model pipeline

if \_\_name\_\_ == "\_\_main\_\_":

ticker = "TATAMOTORS.BO" # Stock ticker symbol

start\_date = "2020-01-01" # Start date

end\_date = "2025-01-01" # End date

stock\_data = load\_stock\_data(ticker, start\_date, end\_date) # Download data

stock\_data = add\_technical\_indicators(stock\_data) # Add features

stock\_data, scaler, close\_scaler = preprocess\_data(stock\_data) # Scale features

features = ['Close', 'Volume', 'RSI', 'Upper\_Band', 'Lower\_Band', 'SMA\_40', 'SMA\_100'] # Selected features

X = stock\_data[features] # Feature matrix

y = stock\_data[['Close']] # Target column

train\_size = int(len(X) \* 0.8) # 80-20 train-test split

X\_train, X\_test = X.iloc[:train\_size], X.iloc[train\_size:] # Split features

y\_train, y\_test = y.iloc[:train\_size], y.iloc[train\_size:] # Split target

rf\_model = train\_random\_forest(X\_train, y\_train.values.ravel()) # Train Random Forest

lstm\_model = train\_lstm(X\_train, y\_train) # Train LSTM

X\_test\_lstm = np.reshape(X\_test.values, (X\_test.shape[0], X\_test.shape[1], 1)) # Reshape test data

rf\_preds = rf\_model.predict(X\_test).reshape(-1, 1) # RF predictions

lstm\_preds = lstm\_model.predict(X\_test\_lstm).reshape(-1, 1) # LSTM predictions

final\_preds = (rf\_preds + lstm\_preds) / 2 # Average predictions from both models

final\_preds = close\_scaler.inverse\_transform(final\_preds) # Convert scaled prices back to actual

mse = mean\_squared\_error(y\_test, final\_preds) # Compute MSE

rmse = np.sqrt(mse) # Compute RMSE

print(f"Mean Squared Error (MSE): {mse}")

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

visualize\_predictions(y\_test, final\_preds, stock\_data['Close']) # Plot predictions

print("Hybrid Model Execution Completed!")