**Hybrid Arima Lstm Rf**

# Import all necessary libraries

import pandas as pd # For data manipulation

import numpy as np # For numerical operations

from datetime import datetime, timedelta # For date operations

from statsmodels.tsa.stattools import adfuller # For stationarity test

from statsmodels.tsa.arima.model import ARIMA # For ARIMA model

from sklearn.ensemble import RandomForestRegressor # For Random Forest regression

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

from sklearn.preprocessing import MinMaxScaler # For scaling inputs

from tensorflow.keras.models import Sequential # Sequential model API

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

import matplotlib.pyplot as plt # For plotting

import yfinance as yf # For downloading stock data

# Function to fetch stock data using yfinance

def fetch\_stock\_data(stock\_ticker, start\_date, end\_date):

stock\_data = yf.download(stock\_ticker, start=start\_date, end=end\_date) # Download data from Yahoo Finance

stock\_data.reset\_index(inplace=True) # Reset index to bring 'Date' as column

stock\_data.columns = [col[0] if isinstance(col, tuple) else col for col in stock\_data.columns] # Clean column names

return stock\_data

# Preprocess stock data (stationarity check, differencing, lag creation)

def preprocess\_data(stock\_data):

stock\_data['Date'] = pd.to\_datetime(stock\_data['Date']).dt.tz\_localize(None) # Format date

if not test\_stationarity(stock\_data['Close']): # Check if Close is stationary

stock\_data['Close\_Diff'] = stock\_data['Close'].diff().dropna() # Apply differencing if not

scaler = MinMaxScaler() # Initialize MinMaxScaler

stock\_data['Lag\_Close'] = scaler.fit\_transform(stock\_data['Close'].values.reshape(-1, 1)) # Scale Close price

stock\_data['Lag\_Close'] = stock\_data['Close'].shift(1) # Create lag feature

stock\_data.dropna(inplace=True) # Drop NaN values

return stock\_data

# Function to test for stationarity using ADF test

def test\_stationarity(series):

result = adfuller(series) # Run ADF test

print("ADF Statistic:", result[0])

print("p-value:", result[1])

return result[1] < 0.05 # Return True if p-value < 0.05 (stationary)

# Define features and target columns

features = ['Lag\_Close']

target = 'Close'

# Forecast using ARIMA model

def arima\_forecast(train, test):

model = ARIMA(train['Close'], order=(5, 1, 0)) # Fit ARIMA with (p,d,q)

model\_fit = model.fit() # Train model

predictions = model\_fit.forecast(steps=len(test)) # Forecast future steps

return predictions

# Forecast using LSTM model

def lstm\_forecast(train, test):

X\_train = np.array(train[features]).reshape((train.shape[0], 1, len(features))) # Reshape input

y\_train = train[target] # Define target

model = Sequential()

model.add(LSTM(50, activation='relu', input\_shape=(1, len(features)))) # LSTM layer

model.add(Dense(1)) # Output layer

model.compile(optimizer='adam', loss='mse') # Compile model

model.fit(X\_train, y\_train, epochs=50, batch\_size=32, verbose=1) # Train model

X\_test = np.array(test[features]).reshape((test.shape[0], 1, len(features))) # Reshape test input

predictions = model.predict(X\_test) # Predict on test data

return predictions.flatten() # Return as flat array

# Forecast using Random Forest Regressor

def random\_forest\_forecast(train, test):

model = RandomForestRegressor(n\_estimators=200, max\_depth=10, random\_state=42) # Initialize model

model.fit(train[features], train[target]) # Train model

predictions = model.predict(test[features]) # Predict on test

return predictions

# Weighted combination of ARIMA, LSTM, and RF predictions

def hybrid\_predictions(arima\_pred, lstm\_pred, rf\_pred, weights):

return (

weights['arima'] \* arima\_pred +

weights['lstm'] \* lstm\_pred +

weights['rf'] \* rf\_pred

)

# Main pipeline function

def main():

stock\_ticker = "TATAMOTORS.BO" # Define stock ticker

start\_date = datetime.now() - timedelta(days=1825) # Start date (5 years ago)

end\_date = datetime.now() # End date (today)

stock\_data = fetch\_stock\_data(stock\_ticker, start\_date, end\_date) # Fetch data

processed\_data = preprocess\_data(stock\_data) # Preprocess it

train\_size = int(len(processed\_data) \* 0.8) # 80% train, 20% test split

train, test = processed\_data.iloc[:train\_size], processed\_data.iloc[train\_size:] # Split data

arima\_pred = arima\_forecast(train, test) # ARIMA forecast

lstm\_pred = lstm\_forecast(train, test) # LSTM forecast

rf\_pred = random\_forest\_forecast(train, test) # RF forecast

weights = {'arima': 0.3, 'lstm': 0.5, 'rf': 0.2} # Weights for each model

final\_pred = hybrid\_predictions(arima\_pred, lstm\_pred, rf\_pred, weights) # Combine forecasts

mse = mean\_squared\_error(test[target], final\_pred) # Compute MSE

rmse = np.sqrt(mse) # Compute RMSE

print(f"RMSE: {rmse}") # Print RMSE

plt.figure(figsize=(12, 6)) # Create plot

plt.plot(test['Date'], test[target], label='Actual Prices', color='blue') # Actual

plt.plot(test['Date'], final\_pred, label='Hybrid Predictions', color='red') # Predicted

plt.xlabel('Date')

plt.ylabel('Stock Price')

plt.title('Hybrid Stock Price Prediction')

plt.legend()

plt.show()

# Entry point for script

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

main()