Ensemble deep learning model

import numpy as np  
import pandas as pd  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  
from keras.models import Sequential  
from keras.layers import Bidirectional, LSTM, SimpleRNN  
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout  
from keras.callbacks import EarlyStopping  
from keras.optimizers import Adam  
from sklearn.preprocessing import MinMaxScaler  
import matplotlib.pyplot as plt  
  
*# Load and preprocess data*df = pd.read\_csv(**"BTC-CoinMarketCap new.csv"**)  
df.drop([**'Date'**], axis=1, inplace=True)  
df.fillna(df.mean(), inplace=True)  
scaler = MinMaxScaler()  
df[df.columns] = scaler.fit\_transform(df[df.columns])  
  
*# Create a function to split data into training and testing sets*def prepare\_data(df, target\_col, test\_size=0.2, window\_len=10):  
 split\_row = int(len(df) \* (1 - test\_size))  
 train\_data = df[:split\_row].copy()  
 test\_data = df[split\_row:].copy()  
  
 X\_train, y\_train = [], []  
 for i in range(len(train\_data) - window\_len):  
 end\_idx = i + window\_len  
 if end\_idx >= len(train\_data):  
 break  
 X\_train.append(train\_data.drop([**'Close'**], axis=1).values[i:end\_idx])  
 y\_train.append(train\_data[target\_col].values[end\_idx])  
 X\_train = np.array(X\_train)  
 y\_train = np.array(y\_train)  
  
 X\_test, y\_test = [], []  
 for i in range(len(test\_data) - window\_len):  
 end\_idx = i + window\_len  
 if end\_idx >= len(test\_data):  
 break  
 X\_test.append(test\_data.drop([**'Close'**], axis=1).values[i:end\_idx])  
 y\_test.append(test\_data[target\_col].values[end\_idx])  
 X\_test = np.array(X\_test)  
 y\_test = np.array(y\_test)  
  
 return X\_train, y\_train, X\_test, y\_test  
  
*# Split data into training and testing sets*window\_len = 10  
X\_train, y\_train, X\_test, y\_test = prepare\_data(df, **'Close'**, test\_size=0.2, window\_len=window\_len)  
  
*# Build and compile LSTM model*model\_lstm = Sequential()  
model\_lstm.add(Bidirectional(LSTM(128, input\_shape=(window\_len, X\_train.shape[2]), return\_sequences=True)))  
model\_lstm.add(Dropout(0.2))  
model\_lstm.add(Bidirectional(LSTM(64, return\_sequences=True)))  
model\_lstm.add(Dropout(0.2))  
model\_lstm.add(Bidirectional(LSTM(32)))  
model\_lstm.add(Dropout(0.2))  
model\_lstm.add(Dense(1))  
model\_lstm.compile(loss=**'mean\_squared\_error'**, optimizer=Adam(learning\_rate=0.01))  
  
*# Define early stopping callback for LSTM*early\_stopping\_lstm = EarlyStopping(monitor=**'val\_loss'**, patience=10, restore\_best\_weights=True)  
  
*# Train the LSTM model*history\_lstm = model\_lstm.fit(X\_train, y\_train, epochs=30, batch\_size=64, validation\_split=0.1, callbacks=[early\_stopping\_lstm])  
  
*# Make predictions on test data using LSTM*y\_pred\_lstm = model\_lstm.predict(X\_test)  
  
*# Convert y\_test and y\_pred to 1D arrays for LSTM*y\_test\_lstm = y\_test.flatten()  
y\_pred\_lstm = y\_pred\_lstm.flatten()  
  
*# Remove NaN values for LSTM*nan\_indices\_lstm = np.isnan(y\_test\_lstm) | np.isnan(y\_pred\_lstm)  
y\_test\_lstm = y\_test\_lstm[~nan\_indices\_lstm]  
y\_pred\_lstm = y\_pred\_lstm[~nan\_indices\_lstm]  
  
*# Build and compile CNN model*model\_cnn = Sequential()  
model\_cnn.add(Conv1D(filters=64, kernel\_size=3, activation=**'relu'**, input\_shape=(window\_len, X\_train.shape[2])))  
model\_cnn.add(MaxPooling1D(pool\_size=2))  
model\_cnn.add(Flatten())  
model\_cnn.add(Dense(128, activation=**'relu'**))  
model\_cnn.add(Dropout(0.2))  
model\_cnn.add(Dense(1))  
model\_cnn.compile(loss=**'mean\_squared\_error'**, optimizer=Adam(learning\_rate=0.01))  
  
*# Define early stopping callback for CNN*early\_stopping\_cnn = EarlyStopping(monitor=**'val\_loss'**, patience=30, restore\_best\_weights=True)  
  
*# Train the CNN model*history\_cnn = model\_cnn.fit(X\_train, y\_train, epochs=100, batch\_size=64, validation\_split=0.1, callbacks=[early\_stopping\_cnn])  
  
*# Make predictions on test data using CNN*y\_pred\_cnn = model\_cnn.predict(X\_test)  
  
*# Convert y\_test and y\_pred to 1D arrays for CNN*y\_test\_cnn = y\_test.flatten()  
y\_pred\_cnn = y\_pred\_cnn.flatten()  
  
*# Remove NaN values for CNN*nan\_indices\_cnn = np.isnan(y\_test\_cnn) | np.isnan(y\_pred\_cnn)  
y\_test\_cnn = y\_test\_cnn[~nan\_indices\_cnn]  
y\_pred\_cnn = y\_pred\_cnn[~nan\_indices\_cnn]  
  
*# Combine predictions using ensemble (simple averaging)*y\_pred\_ensemble = (y\_pred\_lstm + y\_pred\_cnn) / 2  
  
*# Load the actual target values for evaluation*y\_actual = df[**'Close'**].values  
y\_actual = y\_test\_cnn *# Replace with your actual target values  
  
# Evaluate the ensemble model*MAE\_ensemble = mean\_absolute\_error(y\_actual, y\_pred\_ensemble)  
RMSE\_ensemble = mean\_squared\_error(y\_actual, y\_pred\_ensemble, squared=False)  
accuracy\_ensemble = 100 - (MAE\_ensemble + RMSE\_ensemble) \* 100  
  
*# Print evaluation metrics for the ensemble*print(**"MAE (Ensemble):"**, MAE\_ensemble)  
print(**"RMSE (Ensemble):"**, RMSE\_ensemble)  
print(**"Accuracy (Ensemble):"**, accuracy\_ensemble, **"%"**)  
  
*# Plot actual vs. predicted values*plt.figure(figsize=(10, 6))  
plt.plot(y\_actual, label=**'Actual Close Price'**)  
plt.plot(y\_pred\_ensemble, label=**'Predicted Close Price (Ensemble)'**)  
plt.xlabel(**'Time'**)  
plt.ylabel(**'Close Price'**)  
plt.title(**'Actual vs. Predicted Close Prices (Ensemble)'**)  
plt.legend()  
plt.show()