**BTC PREDICTION WITH BI-LSTM**

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  
from keras.models import Sequential  
from keras.layers import LSTM, Dense, Dropout, Bidirectional  
from keras.callbacks import EarlyStopping  
from keras.optimizers import Adam  
from sklearn.preprocessing import MinMaxScaler  
  
*# Set random seed for reproducibility  
#seed\_value = 42  
#np.random.seed(seed\_value)  
  
# Load data*df = pd.read\_csv(**"BTC\_2014\_21.csv"**)  
  
*# Remove unwanted columns*df.drop([**'Volume'**, **'High'**, **'Low'**, **'Adj Close'**], axis=1, inplace=True)  
  
*# Convert 'Date' column to datetime and numeric representation*df[**'Date'**] = pd.to\_datetime(df[**'Date'**], format=**'%m/%d/%Y'**)  
df[**'NumericDate'**] = (df[**'Date'**] - df[**'Date'**].min()).dt.days  
  
*# Normalize data*scaler = MinMaxScaler()  
df[[**'Open'**, **'Close'**, **'NumericDate'**]] = scaler.fit\_transform(df[[**'Open'**, **'Close'**, **'NumericDate'**]])  
  
*# 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[[**'Open'**, **'NumericDate'**]].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[[**'Open'**, **'NumericDate'**]].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 BI-LSTM model*model = Sequential()  
model.add(Bidirectional(LSTM(128, input\_shape=(window\_len, 2), return\_sequences=True)))  
model.add(Dropout(0.2))  
model.add(Bidirectional(LSTM(64, return\_sequences=True)))  
model.add(Dropout(0.2))  
model.add(Bidirectional(LSTM(32)))  
model.add(Dropout(0.2))  
model.add(Dense(1))  
model.compile(loss=**'mean\_squared\_error'**, optimizer=Adam(learning\_rate=0.01))  
  
*# Define early stopping callback*early\_stopping = EarlyStopping(monitor=**'val\_loss'**, patience=10, restore\_best\_weights=True)  
  
*# Train the model*history = model.fit(X\_train, y\_train, epochs=100, batch\_size=64, validation\_split=0.1, callbacks=[early\_stopping])  
  
*# Print model summary*model.summary()  
  
*# Make predictions on test data*y\_pred = model.predict(X\_test)  
  
*# Convert y\_test and y\_pred to 1D arrays*y\_test = y\_test.flatten()  
y\_pred = y\_pred.flatten()  
  
*# Print the number of NaN values*print(np.isnan(y\_test).sum())  
print(np.isnan(y\_pred).sum())  
  
*# Remove NaN values*nan\_indices = np.isnan(y\_test) | np.isnan(y\_pred)  
y\_test = y\_test[~nan\_indices]  
y\_pred = y\_pred[~nan\_indices]  
  
*# Plot actual vs predicted values*plt.figure(figsize=(12, 8))  
plt.plot(y\_test, label=**'Actual'**, linewidth=2, linestyle=**'-'**)  
plt.plot(y\_pred, label=**'Predicted'**, linewidth=2, linestyle=**'-'**)  
plt.xlabel(**'Time'**)  
plt.ylabel(**'Normalized Closing Price'**)  
plt.title(**'Actual vs Predicted Closing Prices'**)  
plt.legend()  
plt.show()  
  
*# Calculate evaluation metrics*MAE = mean\_absolute\_error(y\_test, y\_pred)  
RMSE = mean\_squared\_error(y\_test, y\_pred, squared=False)  
accuracy = 100 - (MAE + RMSE) \* 100  
  
*# Calculate MAPE*def mean\_absolute\_percentage\_error(y\_true, y\_pred):  
 y\_true = np.array(y\_true)  
 y\_pred = np.array(y\_pred)  
 return np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100  
MAPE = mean\_absolute\_percentage\_error(y\_test, y\_pred)  
  
*# Print evaluation metrics*print(**"MAPE:"**, MAPE)  
print(**"MAE:"**, MAE)  
print(**"RMSE:"**, RMSE)  
print(**"Accuracy:"**, accuracy, **"%"**)

**BTC PRECITION WITH LSTM 2014-21**

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  
from keras.models import Sequential  
from keras.layers import LSTM, Dense, Dropout  
from keras.callbacks import EarlyStopping  
from keras.optimizers import Adam  
from sklearn.preprocessing import MinMaxScaler  
  
*# Set random seed for reproducibility  
#seed\_value = 42  
#np.random.seed(seed\_value)  
  
# Load data*df = pd.read\_csv(**"BTC\_2014\_21.csv"**)  
  
*# Remove unwanted columns*df.drop([**'Volume'**, **'High'**, **'Low'**, **'Adj Close'**], axis=1, inplace=True)  
  
*# Convert 'Date' column to datetime and numeric representation*df[**'Date'**] = pd.to\_datetime(df[**'Date'**], format=**'%m/%d/%Y'**)  
df[**'NumericDate'**] = (df[**'Date'**] - df[**'Date'**].min()).dt.days  
  
*# Normalize data*scaler = MinMaxScaler()  
df[[**'Open'**, **'Close'**, **'NumericDate'**]] = scaler.fit\_transform(df[[**'Open'**, **'Close'**, **'NumericDate'**]])  
  
*# 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[[**'Open'**, **'NumericDate'**]].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[[**'Open'**, **'NumericDate'**]].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  
*#window\_len = 20*X\_train, y\_train, X\_test, y\_test = prepare\_data(df, **'Close'**, test\_size=0.2, window\_len=window\_len)  
  
*# Build LSTM model  
#model = Sequential()  
#model.add(LSTM(128, input\_shape=(window\_len, 2), return\_sequences=True))  
#model.add(Dropout(0.2))  
#model.add(LSTM(64))  
#model.add(Dropout(0.2))  
#model.add(Dense(1))  
#model.compile(loss='mean\_squared\_error', optimizer=Adam(learning\_rate=0.01))  
#model.summary()  
  
# Build LSTM model*model = Sequential()  
model.add(LSTM(128, input\_shape=(window\_len, 2), return\_sequences=True))  
model.add(Dropout(0.2))  
model.add(LSTM(64, return\_sequences=True))  
model.add(Dropout(0.2))  
model.add(LSTM(32)) *# Additional LSTM layer*model.add(Dropout(0.2))  
model.add(Dense(1))  
model.compile(loss=**'mean\_squared\_error'**, optimizer=Adam(learning\_rate=0.01))  
model.summary()  
  
  
*# Define early stopping callback  
#early\_stopping = EarlyStopping(monitor='val\_loss', patience=20, restore\_best\_weights=True)  
  
# Train the model  
#history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_split=0.1, callbacks=[early\_stopping])  
  
# Define early stopping callback*early\_stopping = EarlyStopping(monitor=**'val\_loss'**, patience=10, restore\_best\_weights=True)  
  
*# Train the model  
#history = model.fit(X\_train, y\_train, epochs=100, batch\_size=64, validation\_split=0.1, callbacks=[early\_stopping])  
# Make predictions on test data  
#y\_pred = model.predict(X\_test)  
# Train the model*history = model.fit(X\_train, y\_train, epochs=100, batch\_size=64, validation\_split=0.1, callbacks=[early\_stopping])  
  
*# Make predictions on test data*y\_pred = model.predict(X\_test)  
  
*# Make predictions on test data  
#y\_pred = model.predict(X\_test)  
  
# Convert y\_test and y\_pred to 1D arrays*y\_test = y\_test.flatten()  
y\_pred = y\_pred.flatten()  
  
*# Print the number of NaN values*print(np.isnan(y\_test).sum())  
print(np.isnan(y\_pred).sum())  
  
*# Remove NaN values*nan\_indices = np.isnan(y\_test) | np.isnan(y\_pred)  
y\_test = y\_test[~nan\_indices]  
y\_pred = y\_pred[~nan\_indices]  
  
  
*# Plot actual vs predicted values*plt.figure(figsize=(12, 8))  
plt.plot(y\_test, label=**'Actual'**, linewidth=2, linestyle=**'-'**)  
plt.plot(y\_pred, label=**'Predicted'**, linewidth=2, linestyle=**'-'**)  
plt.xlabel(**'Time'**)  
plt.ylabel(**'Normalized Closing Price'**)  
plt.title(**'Actual vs Predicted Closing Prices'**)  
plt.legend()  
plt.show()  
  
*# Calculate evaluation metrics*MAE = mean\_absolute\_error(y\_test, y\_pred)  
RMSE = mean\_squared\_error(y\_test, y\_pred, squared=False)  
accuracy = 100 - (MAE + RMSE) \* 100  
  
def mean\_absolute\_percentage\_error(y\_true, y\_pred):  
 y\_true = np.array(y\_true)  
 y\_pred = np.array(y\_pred)  
 return np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100  
  
*# Calculate MAPE*MAPE = mean\_absolute\_percentage\_error(y\_test, y\_pred)  
print(**"MAPE:"**, MAPE)  
*# Print evaluation metrics*print(**"MAE:"**, MAE)  
print(**"RMSE:"**, RMSE)  
print(**"Accuracy:"**, accuracy, **"%"**)

**ETH\_PRED\_LSTM**

**Ethereum prediction from 2017-22 with LSTM**

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  
from keras.models import Sequential  
from keras.layers import LSTM, Dense, Dropout  
from keras.callbacks import EarlyStopping  
from keras.optimizers import Adam  
from sklearn.preprocessing import MinMaxScaler  
  
*# Set random seed for reproducibility  
#seed\_value = 42  
#np.random.seed(seed\_value)  
  
# Load data*df = pd.read\_csv(**"ETH-USD.csv"**)  
  
*# Remove unwanted columns*df.drop([**'Volume'**, **'High'**, **'Low'**, **'Adj Close'**], axis=1, inplace=True)  
  
*# Convert 'Date' column to datetime and numeric representation*df[**'Date'**] = pd.to\_datetime(df[**'Date'**])  
  
  
df[**'NumericDate'**] = (df[**'Date'**] - df[**'Date'**].min()).dt.days  
  
*# Normalize data*scaler = MinMaxScaler()  
df[[**'Open'**, **'Close'**, **'NumericDate'**]] = scaler.fit\_transform(df[[**'Open'**, **'Close'**, **'NumericDate'**]])  
  
*# 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[[**'Open'**, **'NumericDate'**]].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[[**'Open'**, **'NumericDate'**]].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  
*#window\_len = 20*X\_train, y\_train, X\_test, y\_test = prepare\_data(df, **'Close'**, test\_size=0.2, window\_len=window\_len)  
  
  
*# Build LSTM model*model = Sequential()  
model.add(LSTM(128, input\_shape=(window\_len, 2), return\_sequences=True))  
model.add(Dropout(0.2))  
model.add(LSTM(64, return\_sequences=True))  
model.add(Dropout(0.2))  
model.add(LSTM(32)) *# Additional LSTM layer*model.add(Dropout(0.2))  
model.add(Dense(1))  
model.compile(loss=**'mean\_squared\_error'**, optimizer=Adam(learning\_rate=0.01))  
model.summary()  
  
  
  
*# Define early stopping callback*early\_stopping = EarlyStopping(monitor=**'val\_loss'**, patience=10, restore\_best\_weights=True)  
  
*# Train the model  
#history = model.fit(X\_train, y\_train, epochs=100, batch\_size=64, validation\_split=0.1, callbacks=[early\_stopping])  
# Make predictions on test data  
#y\_pred = model.predict(X\_test)  
# Train the model*history = model.fit(X\_train, y\_train, epochs=100, batch\_size=64, validation\_split=0.1, callbacks=[early\_stopping])  
  
*# Make predictions on test data*y\_pred = model.predict(X\_test)  
  
*# Make predictions on test data  
#y\_pred = model.predict(X\_test)  
  
# Convert y\_test and y\_pred to 1D arrays*y\_test = y\_test.flatten()  
y\_pred = y\_pred.flatten()  
  
*# Print the number of NaN values*print(np.isnan(y\_test).sum())  
print(np.isnan(y\_pred).sum())  
  
*# Remove NaN values*nan\_indices = np.isnan(y\_test) | np.isnan(y\_pred)  
y\_test = y\_test[~nan\_indices]  
y\_pred = y\_pred[~nan\_indices]  
  
  
*# Plot actual vs predicted values*plt.figure(figsize=(12, 8))  
plt.plot(y\_test, label=**'Actual'**, linewidth=2, linestyle=**'-'**)  
plt.plot(y\_pred, label=**'Predicted'**, linewidth=2, linestyle=**'-'**)  
plt.xlabel(**'Time'**)  
plt.ylabel(**'Normalized Closing Price'**)  
plt.title(**'Actual vs Predicted Closing Prices'**)  
plt.legend()  
plt.show()  
  
*# Calculate evaluation metrics*MAE = mean\_absolute\_error(y\_test, y\_pred)  
RMSE = mean\_squared\_error(y\_test, y\_pred, squared=False)  
accuracy = 100 - (MAE + RMSE) \* 100  
  
def mean\_absolute\_percentage\_error(y\_true, y\_pred):  
 y\_true = np.array(y\_true)  
 y\_pred = np.array(y\_pred)  
 return np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100  
  
*# Calculate MAPE*MAPE = mean\_absolute\_percentage\_error(y\_test, y\_pred)  
print(**"MAPE:"**, MAPE)  
*# Print evaluation metrics*print(**"MAE:"**, MAE)  
print(**"RMSE:"**, RMSE)  
print(**"Accuracy:"**, accuracy, **"%"**)

**ETH\_PRED\_BI\_LSTM**

**Ethereum prediction from 2017-22 with Bi\_LSTM**

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  
from keras.models import Sequential  
from keras.layers import LSTM, Dense, Dropout, Bidirectional  
from keras.callbacks import EarlyStopping  
from keras.optimizers import Adam  
from sklearn.preprocessing import MinMaxScaler  
  
*# Set random seed for reproducibility  
#seed\_value = 42  
#np.random.seed(seed\_value)  
  
# Load data*df = pd.read\_csv(**"ETH-USD.csv"**)  
  
*# Remove unwanted columns*df.drop([**'Volume'**, **'High'**, **'Low'**, **'Adj Close'**], axis=1, inplace=True)  
  
*# Convert 'Date' column to datetime and numeric representation*df[**'Date'**] = pd.to\_datetime(df[**'Date'**])  
  
  
df[**'NumericDate'**] = (df[**'Date'**] - df[**'Date'**].min()).dt.days  
*# Normalize data*scaler = MinMaxScaler()  
df[[**'Open'**, **'Close'**, **'NumericDate'**]] = scaler.fit\_transform(df[[**'Open'**, **'Close'**, **'NumericDate'**]])  
  
*# 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[[**'Open'**, **'NumericDate'**]].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[[**'Open'**, **'NumericDate'**]].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 BI-LSTM model*model = Sequential()  
model.add(Bidirectional(LSTM(128, input\_shape=(window\_len, 2), return\_sequences=True)))  
model.add(Dropout(0.2))  
model.add(Bidirectional(LSTM(64, return\_sequences=True)))  
model.add(Dropout(0.2))  
model.add(Bidirectional(LSTM(32)))  
model.add(Dropout(0.2))  
model.add(Dense(1))  
model.compile(loss=**'mean\_squared\_error'**, optimizer=Adam(learning\_rate=0.01))  
  
*# Define early stopping callback*early\_stopping = EarlyStopping(monitor=**'val\_loss'**, patience=10, restore\_best\_weights=True)  
  
*# Train the model*history = model.fit(X\_train, y\_train, epochs=100, batch\_size=64, validation\_split=0.1, callbacks=[early\_stopping])  
  
*# Print model summary*model.summary()  
  
*# Make predictions on test data*y\_pred = model.predict(X\_test)  
  
*# Convert y\_test and y\_pred to 1D arrays*y\_test = y\_test.flatten()  
y\_pred = y\_pred.flatten()  
  
*# Print the number of NaN values*print(np.isnan(y\_test).sum())  
print(np.isnan(y\_pred).sum())  
  
*# Remove NaN values*nan\_indices = np.isnan(y\_test) | np.isnan(y\_pred)  
y\_test = y\_test[~nan\_indices]  
y\_pred = y\_pred[~nan\_indices]  
  
*# Plot actual vs predicted values*plt.figure(figsize=(12, 8))  
plt.plot(y\_test, label=**'Actual'**, linewidth=2, linestyle=**'-'**)  
plt.plot(y\_pred, label=**'Predicted'**, linewidth=2, linestyle=**'-'**)  
plt.xlabel(**'Time'**)  
plt.ylabel(**'Normalized Closing Price'**)  
plt.title(**'Actual vs Predicted Closing Prices'**)  
plt.legend()  
plt.show()  
  
*# Calculate evaluation metrics*MAE = mean\_absolute\_error(y\_test, y\_pred)  
RMSE = mean\_squared\_error(y\_test, y\_pred, squared=False)  
accuracy = 100 - (MAE + RMSE) \* 100  
  
*# Calculate MAPE*def mean\_absolute\_percentage\_error(y\_true, y\_pred):  
 y\_true = np.array(y\_true)  
 y\_pred = np.array(y\_pred)  
 return np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100  
MAPE = mean\_absolute\_percentage\_error(y\_test, y\_pred)  
  
*# Print evaluation metrics*print(**"MAPE:"**, MAPE)  
print(**"MAE:"**, MAE)  
print(**"RMSE:"**, RMSE)  
print(**"Accuracy:"**, accuracy, **"%"**)

**RESULTS**

**LSTM FOR BITCOIN PREDICTION 2014-2021**

MAPE: 7.655277136870242

MAE: 0.06242403891560847

RMSE: 0.07961856693697161

Accuracy: 85.79573941474199 %

**BI-LSTM PREDICTION BTC FROM 2014-21**

MAPE: 10.039145526031064

MAE: 0.07245307244805729

RMSE: 0.0886243292269966

Accuracy: 83.8922598324946 %

**LSTM prediction for ETHEREUM from 2017-22**

MAPE: 11.813609673031127

MAE: 0.043700267350723535

RMSE: 0.05409555669737068

Accuracy: 90.22041759519058 %

**Prediction of ETEREUM with BI-LSTM**

MAPE: 9.743507284063293

MAE: 0.03846355474572647

RMSE: 0.05013252002946906

Accuracy: 91.14039252248045 %