df.isnull().sum()

# Remove rows with missing values

df = df.dropna()

# If you want to remove rows with missing values only for specific columns, you can specify those columns using the subset parameter:

# df\_cleaned = df.dropna(subset=['ColumnName1', 'ColumnName2'])

# The resulting DataFrame df\_cleaned will have the rows with missing values removed.

# visualize the closing price history

plt.figure(figsize=(16,8))

plt.title ('EPS')

plt.plot(df['EPS'])

plt.xlabel('Slot Date', fontsize=18)

plt.ylabel('EPS Rs (₹)', fontsize=18)

plt.show()

# create a new data frame with only the close column

data = df.filter(['EPS'])

#convert the dataframe to a numpy array

dataset = data.values

#Get the number of rows to train the model on

training\_data\_len = math.ceil(len(dataset) \* .8)

training\_data\_len

# Scale the data (preprocessing, normalizing/scaling the data)

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_data = scaler.fit\_transform(dataset)

scaled\_data

#create the training dataset

#create the scaled training dataset

train\_data = scaled\_data[0:training\_data\_len , :]

#split the data into x\_train and y\_train data sets

x\_train = []

y\_train = []

for i in range(60, len(train\_data)):

  x\_train.append(train\_data[i-60:i, 0])

  y\_train.append(train\_data[i,0])

  if i<=60:

    print(x\_train)

    print(y\_train)

    print()

x\_train, y\_train =np.array(x\_train), np.array(y\_train)

# Reshape the data,because LSTM requires inputs to be in 3-dim(no.of samples,no.of time stamps, no.of features) and right now our data is in 2-dim

 #x\_train.shape # we will only get 2-dim (rows,columns)

# now convert the array to 3-dim

x\_train = np.reshape(x\_train,(x\_train.shape[0], x\_train.shape[1], 1)) #no. of rows, no.columns, closing price

x\_train.shape

#Built the LSTM model

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(x\_train.shape[1],1)))

model.add(LSTM(50, return\_sequences= False))

model.add(Dense(25))

model.add(Dense(1))

# compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

#Train the model

model.fit(x\_train,y\_train,batch\_size=1,epochs=5)

#create the testing data set

#create a new array containing scaled values from index 2047 to 2507

test\_data = scaled\_data[training\_data\_len - 60:, :]

#create the data sets x\_test and y\_test

x\_test = []

y\_test = dataset[training\_data\_len:, :] # all the values that we want our model to predict, actual test values

for i in range(60, len(test\_data)):

  x\_test.append(test\_data[i-60:i,0])

  # these values are not scaled, they are the original data

# Convert the data to a numpy array

x\_test = np.array(x\_test)

#Reshape the data (from 2-dim to 3-dim)

x\_test =np.reshape(x\_test,(x\_test.shape[0], x\_test.shape[1],1)) #No.of rows,no.of columns,no.of features

# get the model's predicted price values (we want predicitions based on the x\_test dataset)

predictions = model.predict(x\_test)

predictions = scaler.inverse\_transform(predictions) # we want predictions to contain the same values as our y\_test data set contains

# we are unscaling the values

# get the root mean squared error (RMSE) lower value indicates a better fit

rmse = np.sqrt(np.mean(((predictions- y\_test)\*\*2)))

rmse

# plot the data

train = data[:training\_data\_len]

valid = data[training\_data\_len:]

valid['Predictions'] = predictions

# visualize the data

plt.figure(figsize=(16,8))

plt.title('Model')

plt.xlabel('Data', fontsize=18)

plt.ylabel('EPS INR', fontsize=18)

plt.plot(train['EPS'])

plt.plot(valid[['EPS', 'Predictions']])

plt.legend(['Train','Val','Predictions'], loc='lower right')

plt.show()

valid['Error Godrej'] = ((valid['Predictions'] - valid['EPS']) / valid['EPS']) \* 100

valid

average = valid['Error Godrej'].mean()

# Print the average

print("Average:", average)

# import the libraries

!pip install pandas\_datareader pandas --upgrade

import yfinance as yf

import math

import pandas\_datareader as web

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

from keras.layers import Dense, LSTM

import matplotlib.pyplot as plt

plt.style.use('fivethirtyeight')

#Get the stock code

# Specify the stock symbol

stock\_symbol = 'GODREJPROP.NS'

# Define the date range

start\_date = '2013-01-01'

end\_date = '2023-09-22'

# Fetch the stock data

df = yf.download(stock\_symbol, start=start\_date, end=end\_date)

# Display the data

print(df)

# visualize the closing price history

plt.figure(figsize=(16,8))

plt.title ('Close Price History')

plt.plot(df['Close'])

plt.xlabel('Date', fontsize=18)

plt.ylabel('Close Price Rs (₹)', fontsize=18)

plt.show()

# create a new data frame with only the close column

data = df.filter(['Close'])

#convert the dataframe to a numpy array

dataset = data.values

#Get the number of rows to train the model on

training\_data\_len = math.ceil(len(dataset) \* .8)

training\_data\_len

# Scale the data (preprocessing, normalizing/scaling the data)

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_data = scaler.fit\_transform(dataset)

scaled\_data

#create the training dataset

#create the scaled training dataset

train\_data = scaled\_data[0:training\_data\_len , :]

#split the data into x\_train and y\_train data sets

x\_train = []

y\_train = []

for i in range(60, len(train\_data)):

  x\_train.append(train\_data[i-60:i, 0])

  y\_train.append(train\_data[i,0])

  if i<=60:

    print(x\_train)

    print(y\_train)

    print()

#convert the x\_train and y\_train to numpy array, so that we can now use them to train the model

x\_train, y\_train =np.array(x\_train), np.array(y\_train)

# Reshape the data,because LSTM requires inputs to be in 3-dim(no.of samples,no.of time stamps, no.of features) and right now our data is in 2-dim

 #x\_train.shape # we will only get 2-dim (rows,columns)

# now convert the array to 3-dim

x\_train = np.reshape(x\_train,(x\_train.shape[0], x\_train.shape[1], 1)) #no. of rows, no.columns, closing price

x\_train.shape

#Built the LSTM model

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(x\_train.shape[1],1)))

model.add(LSTM(50, return\_sequences= False))

model.add(Dense(25))

model.add(Dense(1))

# compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

#create the testing data set

#create a new array containing scaled values from index 2047 to 2507

test\_data = scaled\_data[training\_data\_len - 60:, :]

#create the data sets x\_test and y\_test

x\_test = []

y\_test = dataset[training\_data\_len:, :] # all the values that we want our model to predict, actual test values

for i in range(60, len(test\_data)):

  x\_test.append(test\_data[i-60:i,0])

  # these values are not scaled, they are the original data

# Convert the data to a numpy array

x\_test = np.array(x\_test)

#Reshape the data (from 2-dim to 3-dim)

x\_test =np.reshape(x\_test,(x\_test.shape[0], x\_test.shape[1],1)) #No.of rows,no.of columns,no.of features

# get the model's predicted price values (we want predicitions based on the x\_test dataset)

predictions = model.predict(x\_test)

predictions = scaler.inverse\_transform(predictions) # we want predictions to contain the same values as our y\_test data set contains

# we are unscaling the values

# get the root mean squared error (RMSE) lower value indicates a better fit

rmse = np.sqrt(np.mean(((predictions- y\_test)\*\*2)))

rmse

# plot the data

train = data[:training\_data\_len]

valid = data[training\_data\_len:]

valid['Predictions'] = predictions

# visualize the data

plt.figure(figsize=(16,8))

plt.title('Model')

plt.xlabel('Data', fontsize=18)

plt.ylabel('Close Price INR', fontsize=18)

plt.plot(train['Close'])

plt.plot(valid[['Close', 'Predictions']])

plt.legend(['Train','Val','Predictions'], loc='lower right')

plt.show()

# show the valid and predicted prices

valid

valid['Error Godrej'] = ((valid['Predictions'] - valid['Close']) / valid['Close']) \* 100

valid

average = valid['Error Godrej'].mean()

# Print the average

print("Average:", average)