COVID-19 Predictions and Forecasting using Recurrent Neural Networks(RNNs) with Keras

School of Computer and Information Sciences
University of Hyderabad
Gachibowli, Hyderabad - 500 046
Telangana, India

Submitted To:-
Dr. M Nagamani
SCIS, UoH

Submitted By:-
Boja Yashoda Krishna
CSE-3rd year
KGRCET

Table of Contents:-

Abstract:-

COVID-19, responsible for economic inflation across the globe, requires a detailed study of the Data to develop adequate short-term prediction models for forecasting the number of future cases. In this perspective, it is possible to develop strategic planning in the public health system to avoid deaths and reduce spread as well as managing patients. Here, we proposed forecast models comprising long short-term memory (LSTM) assessed for time series prediction of confirmed cases, deaths, and recoveries in 4 major countries affected due to COVID-19. The performance of models is measured by mean absolute error, root means square error. Based on demonstrated robustness and enhanced prediction accuracy, LSTM can be exploited for pandemic prediction for better planning and management.

Keywords: Covid-19,LSTM,RNNs,epidemic prediction,Deep learning models.

Introduction:-

COVID-19 has followed specific patterns and these patterns are based on the dynamic transmission of the epidemic. When it occurs, superseding measures of different methods are used to find and evaluate such infective diseases. Any epidemic in a state or country has arisen with a different aspect of magnitude concerning time, particularly weather period changes and spread of the virus over the period, and exhibited as non-linear. To capture these non-linear compelling changes, researchers have gained attention and designed such non-linear systems to describe the abruptness of infective diseases. Therefore, mathematical models such as SIR (susceptible-infective-removed) for analyzing the epidemics have been introduced. A transmission model with incubation time for malaria and a deterministic model to analyze the interaction between HIV and tuberculosis is successfully developed to solve the nonlinear behavior of parameters. Similar models of discrete-time equations are used to control the infected population.

Deep learning (DL) algorithms show a vital role in the analysis and prediction of huge outbreak data patterns and help in early exploitation to stop the spread rate of coronavirus. COVID-19 is a time-series data and vastly endorsed the use of sequential models to deal with its dynamic nature. In this paper, we proposed the long short term memory (LSTM) to evaluate the predictions with confirmed, negative released, and death cases of COVID-19.

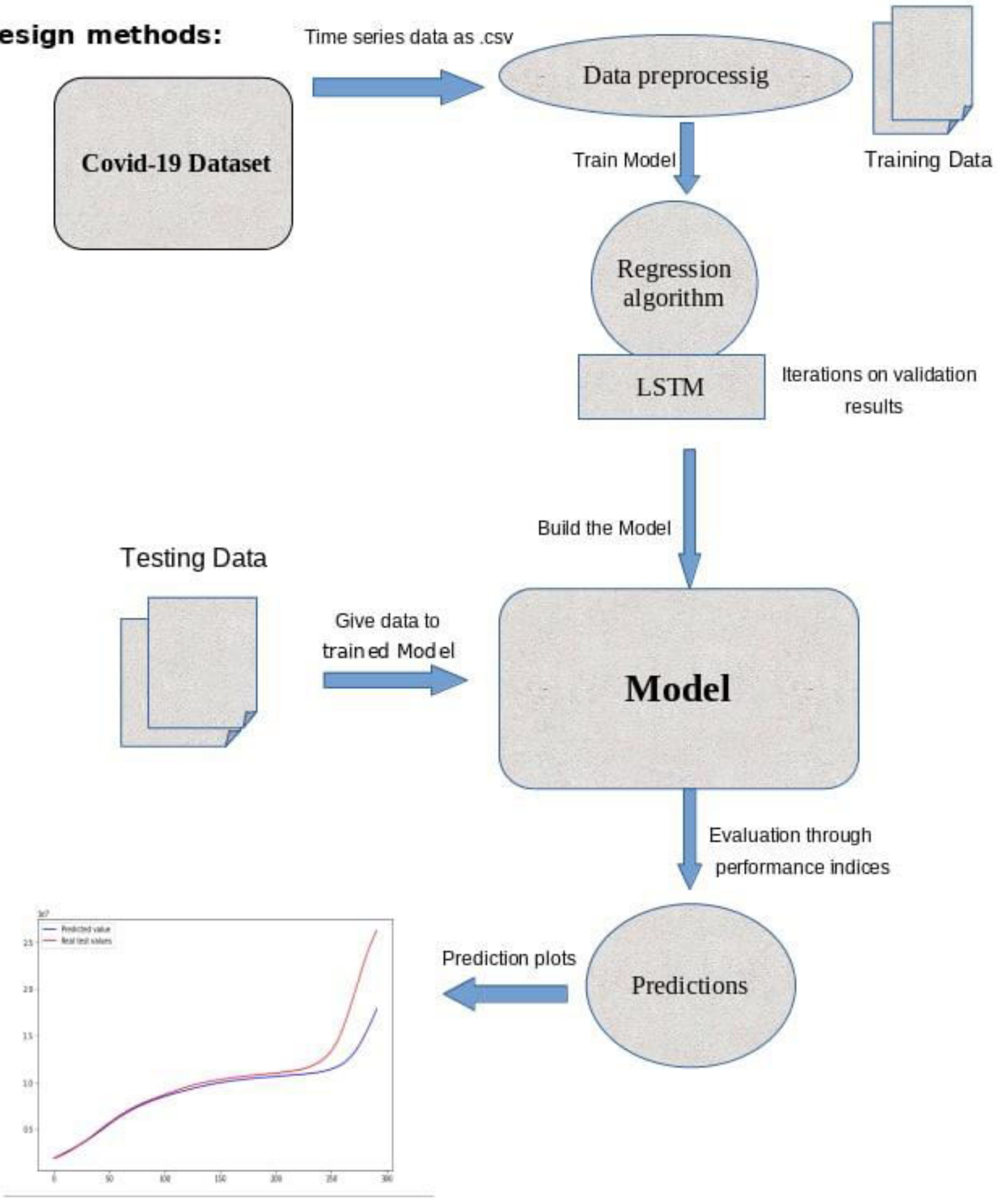
The novelty of the reported work lies in creating the three categories of confirmed cases, death cases, and recovered cases from the dataset and intelligently developing a COVID-19 predictor to predict and analyze future trends of these three categories. This experiment is based on the data set of confirmed COVID-19 cases available until May 21, 2021. Additionally, owing to the dynamic nature of coronavirus, ML and DL models have been implemented for early predictions.

Prominent features of the methodology are

- DL mechanisms of LSTM are proposed to predict the COVID-19 three categories, confirmed cases, deaths, and recovered cases for 4 countries.
- The accuracy of models is measured in terms of MAE

The rest of the report is organized as follows:
Section II describes the proposed methodologies, dataset, and performance metrics, Section III includes detailed results of the designed scheme. While the conclusion is provided in the last section.

Design methods:



r

Methods used:-

LSTM:

RNN has been employed for sequential time series applications with temporal dependencies. An unfolded RNN can process current data by use of previous data. Meanwhile, RNN has the problem to train the long-term dependencies data, which is solved by one of the variants of RNN. LSTM anticipated by Hochreiter and Schmidhuber has been used as an advanced version of the RNN network and has overcome the limitation of RNN by the use of hidden layer units known as memory cells. Memory cells have the self-connections that stored the network temporal state and controlled through three gates named: input gate, output gate, and forget gate. The work of input gate and output gate is used to control the flow of memory cell input and outputs into the rest of the network. In addition, forget gate has been added to the memory cell, which passes the output information with high weights from the previous to the next neuron. The information that resides in memory depends upon the high activation results; if the input unit has high activation, the information is stored in the memory cell. In addition, if the output unit has high activation then it will pass the information to the next neuron. Otherwise, input information with high weights resides in the memory cell.

LSTM network is compute mapping between input sequence and output sequence,*i.e.*X=(X1,X2,...,Xn) andy=(y1,y2,...,yn). Calculating by the following equations:

$$f_{orget}g_{ate} = sigmoid\left(W_{fg}X_t + W_{hfg}h_{t-1} + b_{fg}\right)$$

$$i_{nput}g_{ate} = sigmoid\left(W_{ig}X_t + W_{hig}h_{t-1} + b_{ig}\right)$$

$$o_{utput}g_{ate} = sigmoid\left(W_{og}X_t + W_{hog}h_{t-1} + b_{og}\right)$$

$$\begin{aligned}(C)_t &= (C)_{t-1} \otimes \left(f_{orget}g_{ate}\right)_t + \left(i_{nput}g_{ate}\right)_t \\ &\otimes \left(\tanh\left(W_C X_t + W_{hC}h_{t-1} + b_C\right)\right)\end{aligned}$$

$$h_t = o_{utput}g_{ate} \otimes \tanh\left((C)_{t-1}\right)$$

Wig, Wog, WhC, Wfg and *bfg, big, bog, bC* represent the weights and bias variables respectively of three gates and a memory cell. Here,ht−1symbolizes the prior hidden layers units that element-wise adding with weights of three gates.

To overcome the limitations of the LSTM cell which can work on previous content but cannot use the future one. Schuster and Paliwal proposed bidirectional recurrent neural networks (BRNN) that are comprised of two distinct LSTM hidden layers with similar output in opposite directions. With this architecture, previous and future information is exploited in the output layer. An input sequenceX=(X1,X2,...,Xn) in Bi-LSTM is calculated in forward direction as ht→=(h1,−→h2,−→...,hn−→)and backward directions as ht←=(h1,←−h2,←−...,hn←−). The final out of this cell *yt* is formed by both ht→and ht←, the final sequence of outlooks like y=(y1,y2,...yt...,yn).

Dataset Used:-

1) Indian covid data
Covid-19_INDIA Dataset

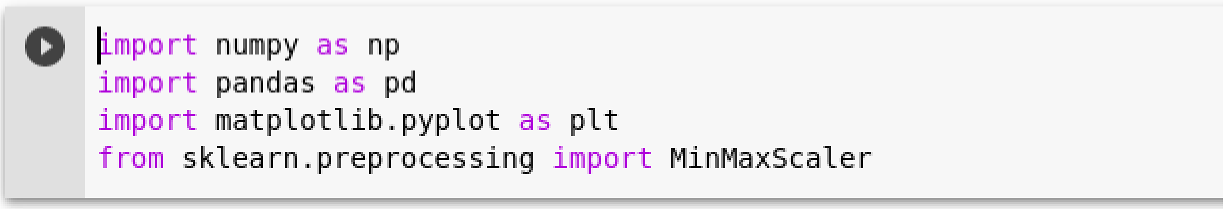
2) World covid data
Covid-19_world Dataset

The .csv file of confirmed cases, death cases, and recovered cases of all countries are provided column-wise. An individual file is created of these three categories from 21 March 2020 to 21 May 2021. Covid19 dataset contains the number of confirmed cases, deaths, and recovered cases and we have taken cases from 21/03/2020 to 01/01/2021 for training purposes and to predict cases from 01/02/2021 to 21/05/2021. For each country, data comprises given cases for 385 days and have to predict for the next 30 days. The data is preprocessed before it is given to ML models for training.

Source Code:-

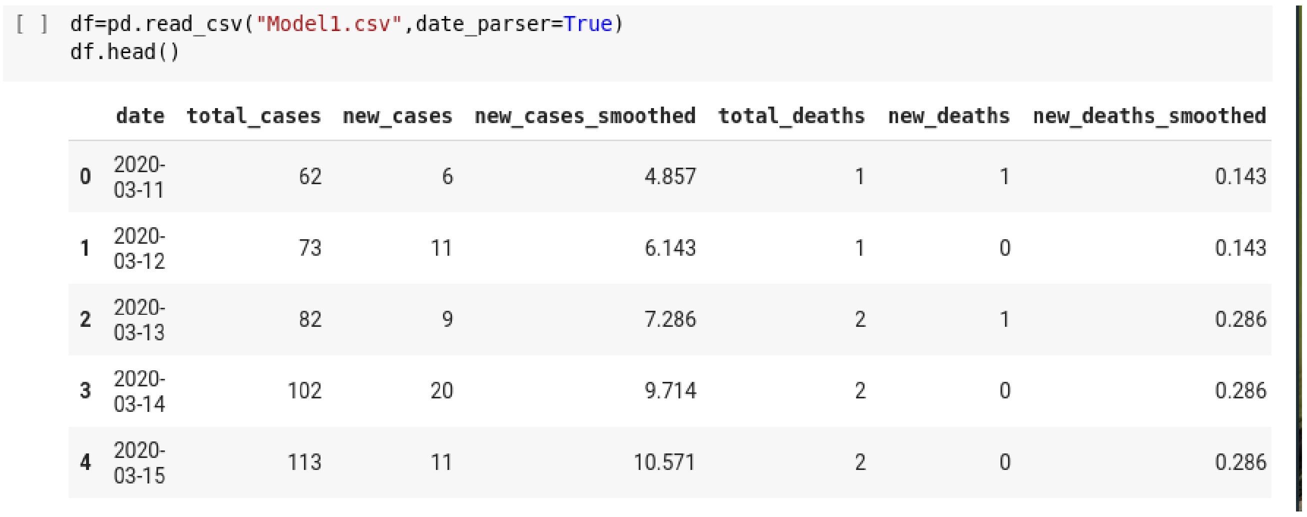
RNN model:

step-1: importing the required libraries



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
```

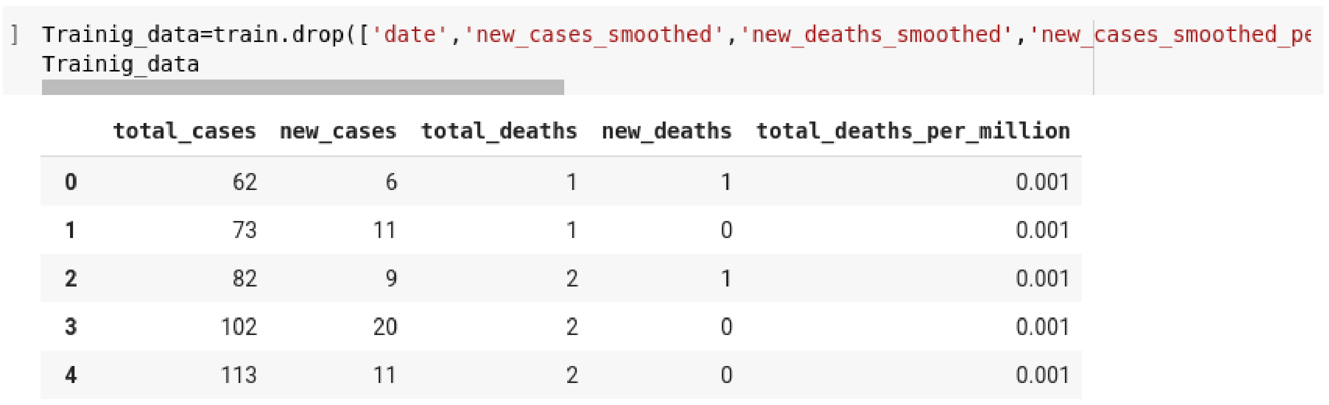
step-2: Invoking the dataset



```
[ ] df=pd.read_csv("Model1.csv",date_parser=True)
    df.head()
```

	date	total_cases	new_cases	new_cases_smoothed	total_deaths	new_deaths	new_deaths_smoothed
0	2020-03-11	62	6	4.857	1	1	0.143
1	2020-03-12	73	11	6.143	1	0	0.143
2	2020-03-13	82	9	7.286	2	1	0.286
3	2020-03-14	102	20	9.714	2	0	0.286
4	2020-03-15	113	11	10.571	2	0	0.286

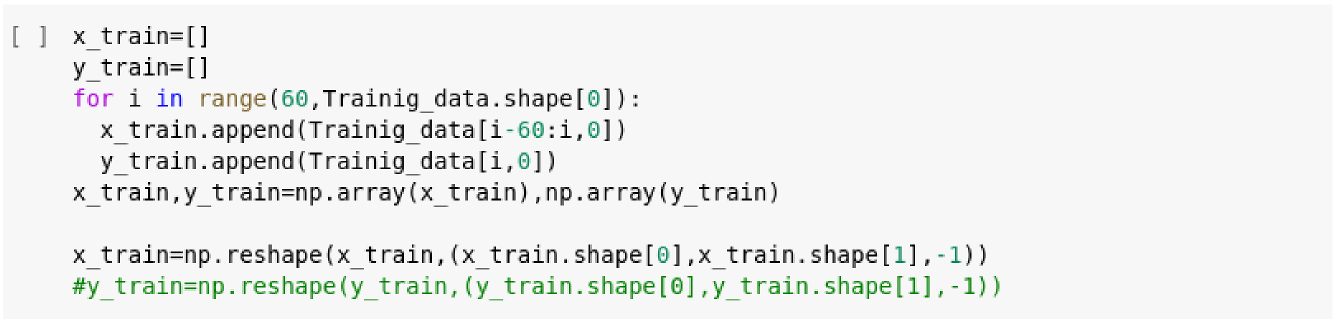
step-3: Preprocessing the dataset



```
] Trainig_data=train.drop(['date','new_cases_smoothed','new_deaths_smoothed','new_cases_smoothed_pe
  Trainig_data
```

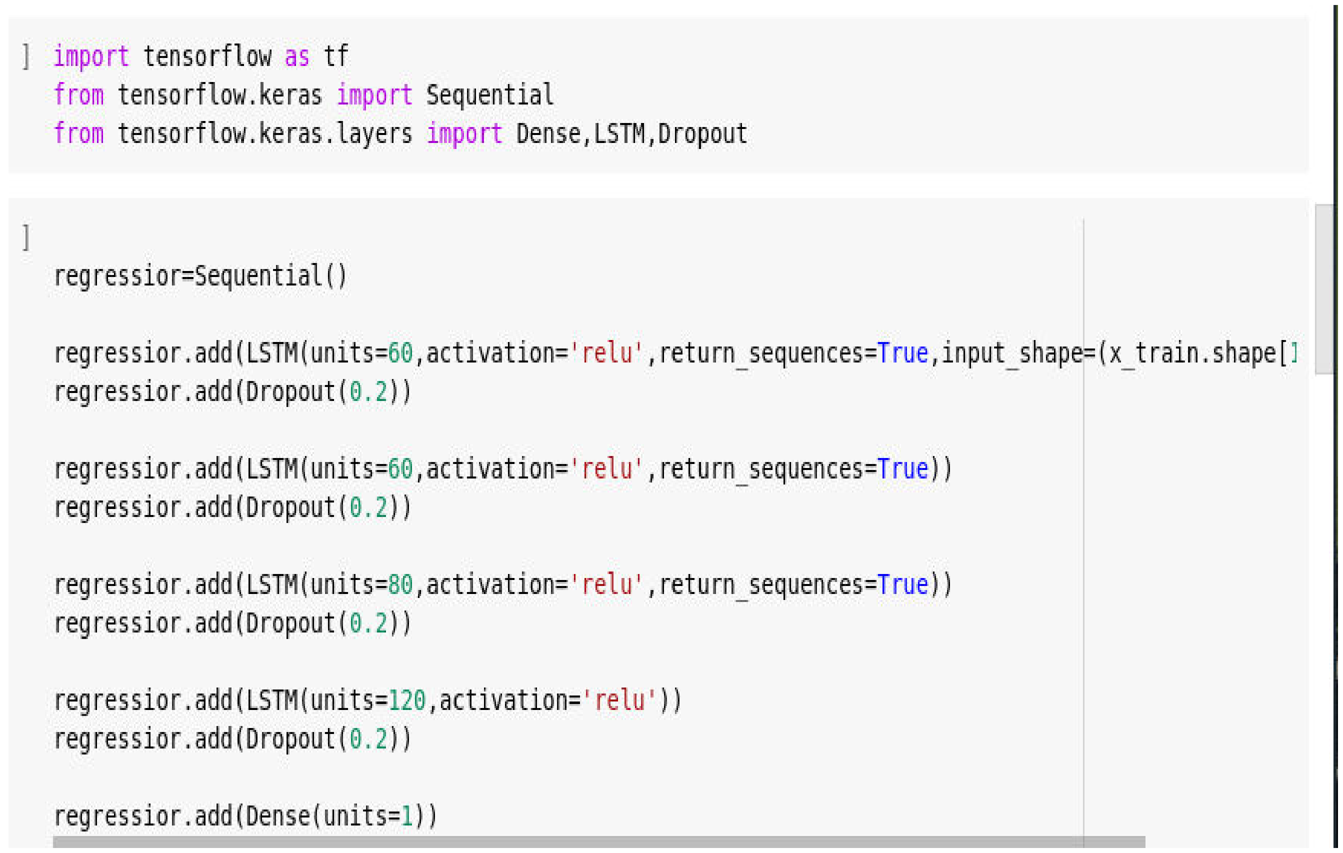
	total_cases	new_cases	total_deaths	new_deaths	total_deaths_per_million
0	62	6	1	1	0.001
1	73	11	1	0	0.001
2	82	9	2	1	0.001
3	102	20	2	0	0.001
4	113	11	2	0	0.001

step-4: Preparing the training data



```
x_train=[]
y_train=[]
for i in range(60,Trainig_data.shape[0]):
  x_train.append(Trainig_data[i-60:i,0])
  y_train.append(Trainig_data[i,0])
x_train,y_train=np.array(x_train),np.array(y_train)

x_train=np.reshape(x_train,(x_train.shape[0],x_train.shape[1],-1))
#y_train=np.reshape(y_train,(y_train.shape[0],y_train.shape[1],-1))
```

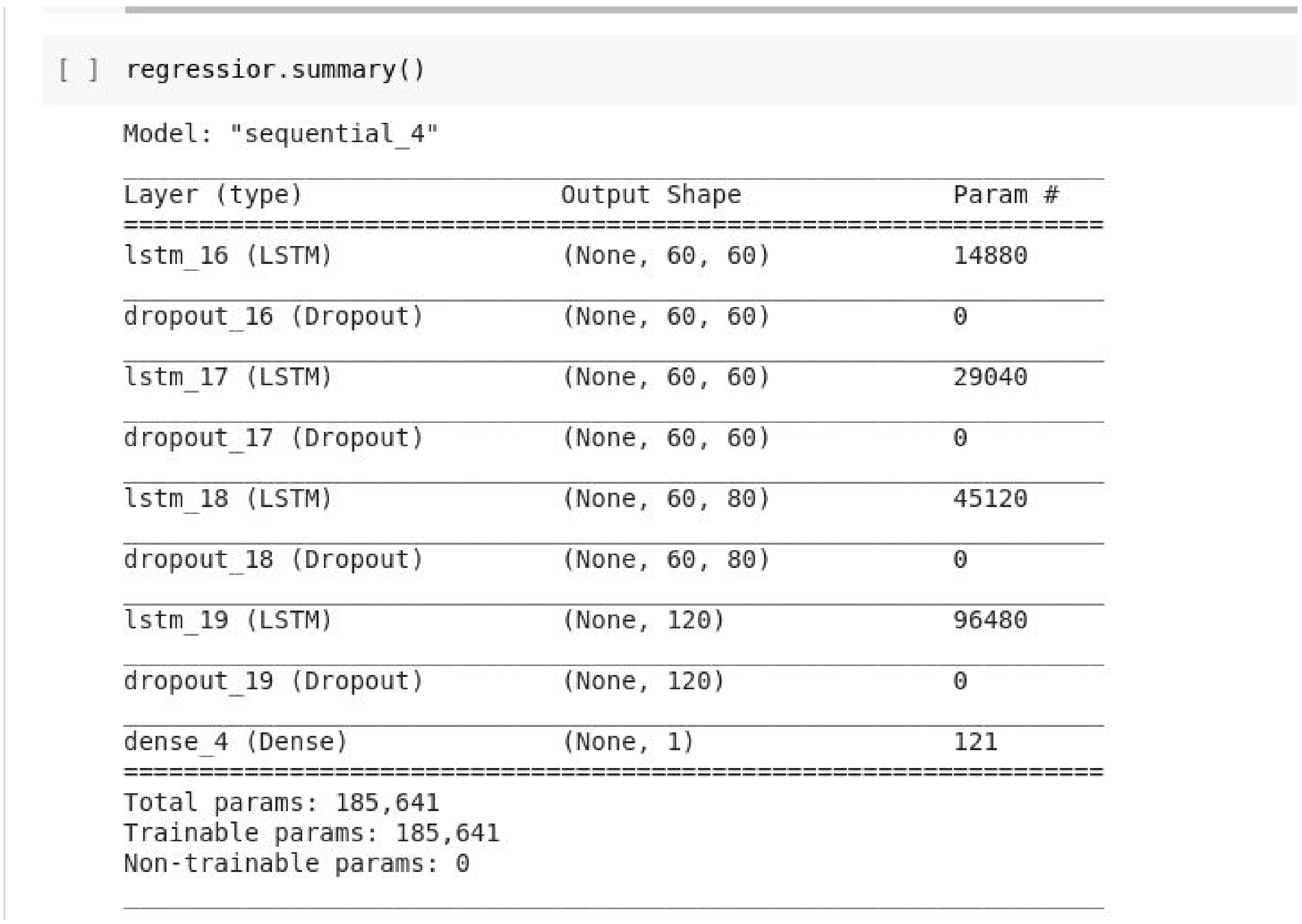
Step-5*:* invoking LSTM, dense, Dropout layers from Keras
and Building the model



```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,LSTM,Dropout
```

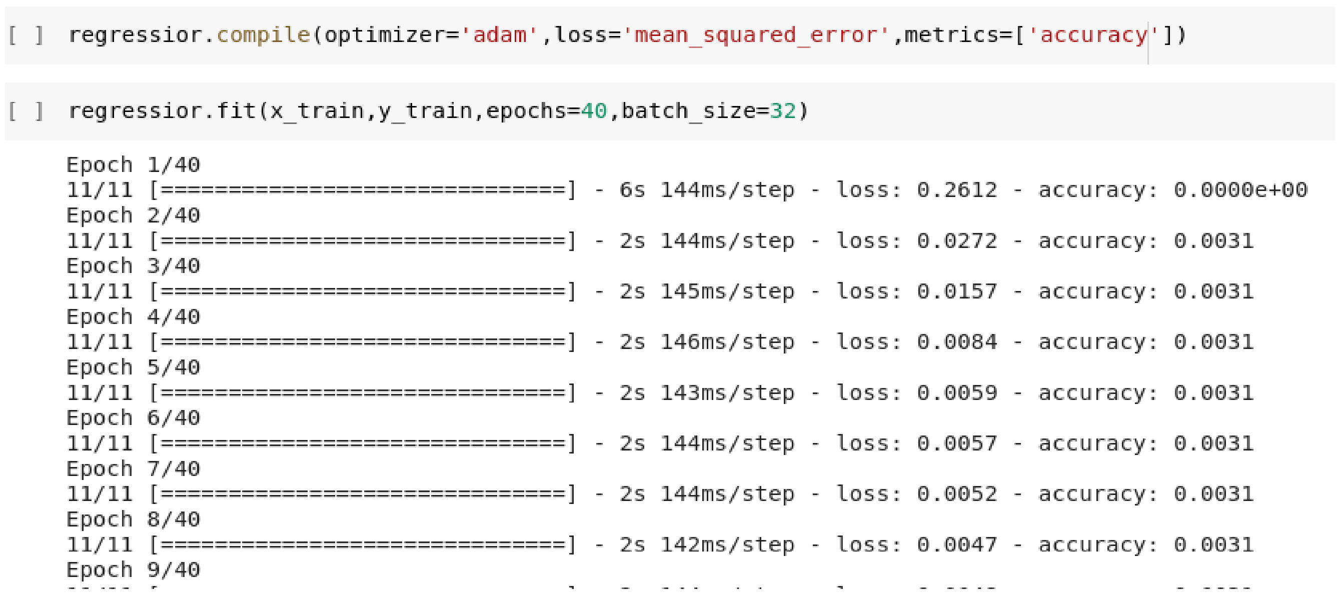
```

regressior=Sequential()

regressior.add(LSTM(units=60,activation='relu',return_sequences=True,input_shape=(x_train.shape[1
regressior.add(Dropout(0.2))

regressior.add(LSTM(units=60,activation='relu',return_sequences=True))
regressior.add(Dropout(0.2))

regressior.add(LSTM(units=80,activation='relu',return_sequences=True))
regressior.add(Dropout(0.2))

regressior.add(LSTM(units=120,activation='relu'))
regressior.add(Dropout(0.2))

regressior.add(Dense(units=1))
```

→Model summary



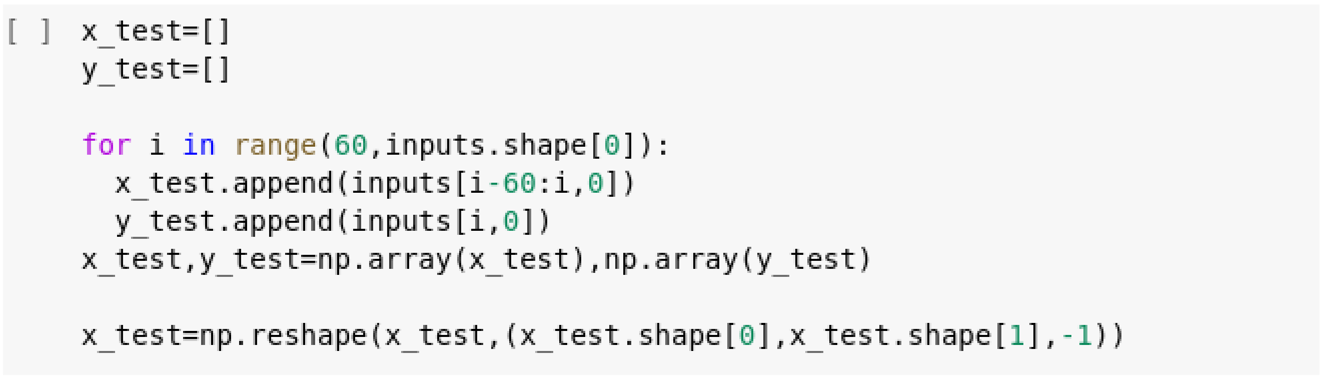
```
[ ]  regressior.summary()

     Model: "sequential_4"
     _________________________________________________________________
     Layer (type)                Output Shape              Param #
     =================================================================
     lstm_16 (LSTM)              (None, 60, 60)            14880
     _________________________________________________________________
     dropout_16 (Dropout)        (None, 60, 60)            0
     _________________________________________________________________
     lstm_17 (LSTM)              (None, 60, 60)            29040
     _________________________________________________________________
     dropout_17 (Dropout)        (None, 60, 60)            0
     _________________________________________________________________
     lstm_18 (LSTM)              (None, 60, 80)            45120
     _________________________________________________________________
     dropout_18 (Dropout)        (None, 60, 80)            0
     _________________________________________________________________
     lstm_19 (LSTM)              (None, 120)               96480
     _________________________________________________________________
     dropout_19 (Dropout)        (None, 120)               0
     _________________________________________________________________
     dense_4 (Dense)             (None, 1)                 121
     =================================================================
     Total params: 185,641
     Trainable params: 185,641
     Non-trainable params: 0
     _________________________________________________________________
```

Step-6: compiling and training the model



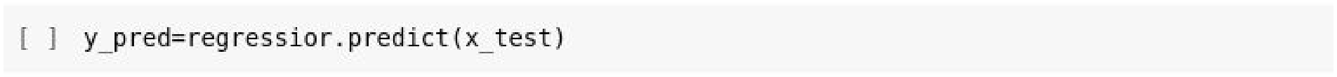
```
[ ]  regressior.compile(optimizer='adam',loss='mean_squared_error',metrics=['accuracy'])

[ ]  regressior.fit(x_train,y_train,epochs=40,batch_size=32)

     Epoch 1/40
     11/11 [==============================] - 6s 144ms/step - loss: 0.2612 - accuracy: 0.0000e+00
     Epoch 2/40
     11/11 [==============================] - 2s 144ms/step - loss: 0.0272 - accuracy: 0.0031
     Epoch 3/40
     11/11 [==============================] - 2s 145ms/step - loss: 0.0157 - accuracy: 0.0031
     Epoch 4/40
     11/11 [==============================] - 2s 146ms/step - loss: 0.0084 - accuracy: 0.0031
     Epoch 5/40
     11/11 [==============================] - 2s 143ms/step - loss: 0.0059 - accuracy: 0.0031
     Epoch 6/40
     11/11 [==============================] - 2s 144ms/step - loss: 0.0057 - accuracy: 0.0031
     Epoch 7/40
     11/11 [==============================] - 2s 144ms/step - loss: 0.0052 - accuracy: 0.0031
     Epoch 8/40
     11/11 [==============================] - 2s 142ms/step - loss: 0.0047 - accuracy: 0.0031
     Epoch 9/40
```

Step-7: Preparing the test data

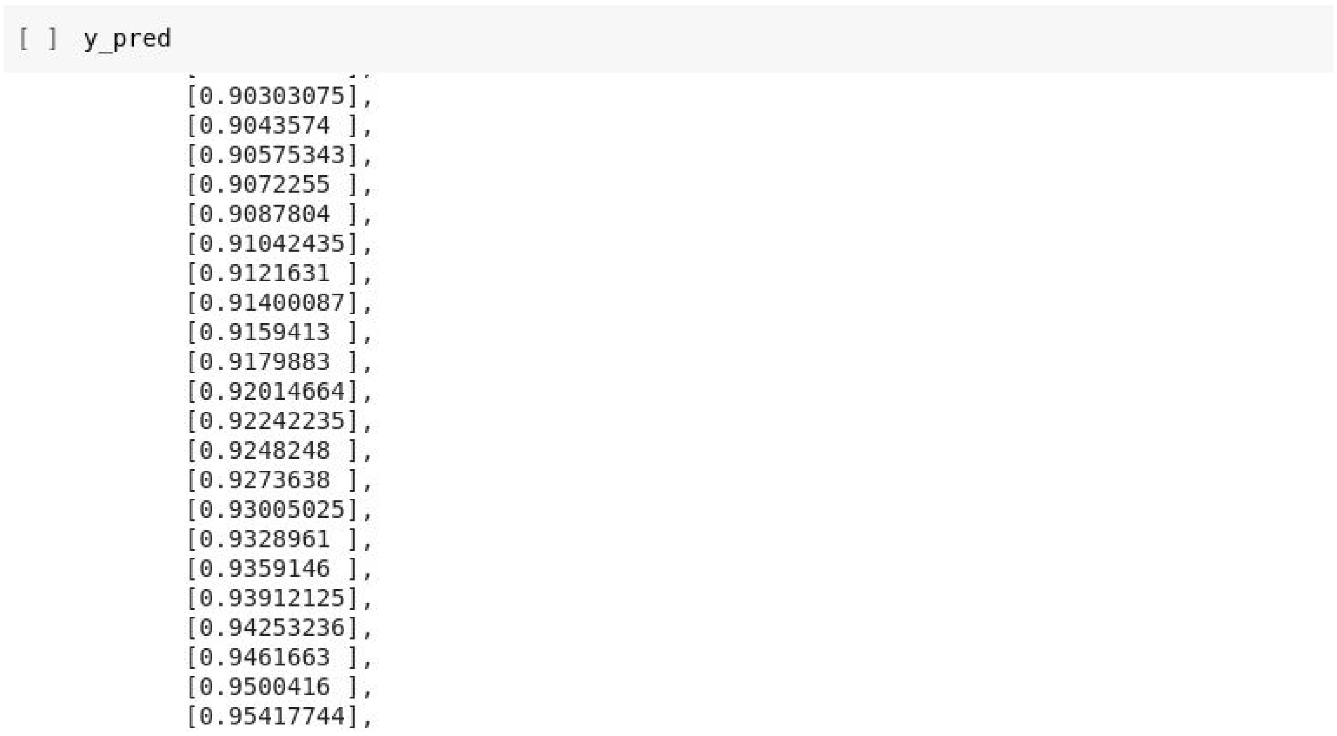


```
[ ]  x_test=[]
     y_test=[]

     for i in range(60,inputs.shape[0]):
       x_test.append(inputs[i-60:i,0])
       y_test.append(inputs[i,0])
     x_test,y_test=np.array(x_test),np.array(y_test)

     x_test=np.reshape(x_test,(x_test.shape[0],x_test.shape[1],-1))
```

Step-8: making the predictions



```
[ ]  y_pred=regressior.predict(x_test)
```



```
[ ]  y_pred
```

```
           [0.90303075],
           [0.9043574 ],
           [0.90575343],
           [0.9072255 ],
           [0.9087804 ],
           [0.91042435],
           [0.9121631 ],
           [0.91400087],
           [0.9159413 ],
           [0.9179883 ],
           [0.92014664],
           [0.92242235],
           [0.9248248 ],
           [0.9273638 ],
           [0.93005025],
           [0.9328961 ],
           [0.9359146 ],
           [0.93912125],
           [0.94253236],
           [0.9461663 ],
           [0.9500416 ],
           [0.95417744],
```

Step-9: plotting predicted value and the real values

```
[ ]  plt.figure(figsize=(10,6))

     plt.plot(y_pred,color='blue',label='Predicted value')
     plt.plot(y_test,color='red',label='Real test values')
     plt.legend()
     plt.show()
```

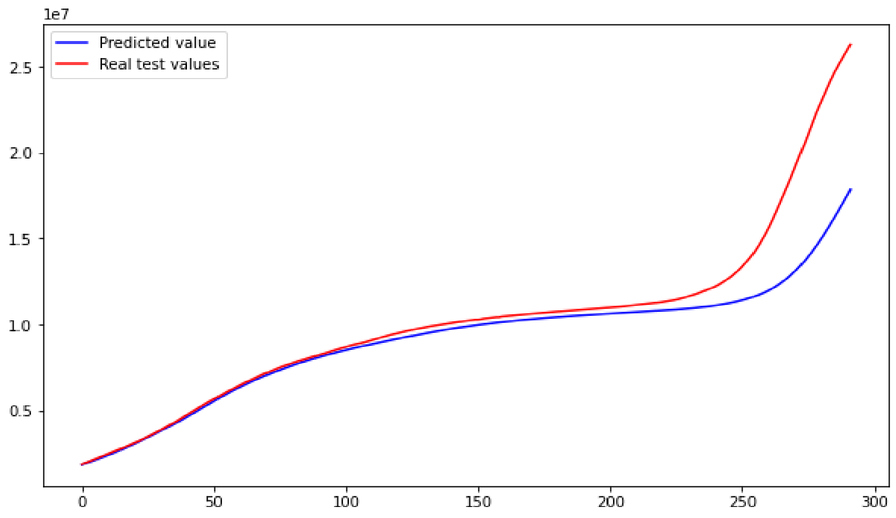
NOTES:
→ RNN model is built with 4 LSTM layers with each containing 50 units and each layer consist of a Dropout of 20% with 1 Dense unit.
→ 'adam' optimizer is used to compile the model with 'mean_squared_error' as a loss function.
→MinMaxScaler is used for scaling the training and testing data.
→ Finally, the predicted values are plotted against the Real values.

Results and Discussion:-

- Proposed scheme with parameters and their values.

LSTM	Layers	3
	No. of neurons	{60,60,80,120}
	Learning rate	0.001
	Optimizer	Adam
	Batch size	32
	Epochs	40
	Activation	relu

Prediction plots:



Conclusion:-

This work studies the effect of the Covid-19 pandemic in India along with 3 other countries. An analytical epidemiological model was developed for the Covid-19 pandemic where model parameters are continuously updated to intelligently adapt to new data sets using an RNN based adaptive online Incremental Learning technique. India was taken as a case study. However, this model can be applied to any population in the world and would be useful in improving decision-making efficiency, policy formulation, monitoring, and forecasting of an epidemic. The model used here is non-intrusive, adaptive, intelligent, and real-time, therefore it is immune to loss of accuracy, reliability, or computational performance arising due to limitations like run-time duration, size of training data, computational complexity.

Inferences on the performance of proposed schemes are listed as follows:

- COVID-19 dataset has been modeled using regressor, LSTM is used for future predictions on confirmed cases, deaths, and recovered cases for 4 countries across the globe.
- Performance measures of MAE, RMSE have been used to evaluate the performance.

It can be concluded that LSTM is an appropriate predictor for such sequential data and capable of predicting with enhanced accuracy for similar other datasets for appropriate planning and better management.

However, the information provided can provide significant insights into the pattern of disease spread and help in making a realistic assessment of the situation necessary for optimal policy framing.

References:-

1)RNN with LSTM layers
https://www.kaggle.com/frlemarchand/covid-19-forecasting-with-an-rnn#4.-Performance-during-training

2)RNN with keras on stock data
https://youtu.be/arydWPLDnEc

3)LSTM with PyTorch
https://www.kaggle.com/kanncaa1/long-short-term-memory-with-pytorch

4)Keras Documentation
https://keras.io/examples/