**Source Code**

**url.py:**

from django.contrib import admin

from django.urls import path

from django.conf.urls.static import static

from django.conf import settings

from farmers import views as farmers

from admn import views as admn

urlpatterns = [

path('admin/', admin.site.urls),

path('index/',farmers.index,name='index'),

# path('lstm/',farmers.lstm,name='lstm'),

path('farmerlogin/',farmers .farmerLogin, name='farmerlogin'),

path('farmerpage/', farmers.farmerpage, name='farmerpage'),

path('farmerregister/', farmers.farmerregister, name='farmerregister'),

path('farmerlogincheck/', farmers.farmerlogincheck, name='farmerlogincheck'),

path('farmerdetails/', farmers.farmerdetails, name='farmerdetails'),

path('activatefarmer/', farmers.activatefarmer, name='activatefarmer'),

path('storecsvdata/', farmers.storecsvdata, name='storecsvdata'),

path('cropdata/', farmers.cropdata,name='cropdata'),

path('cropprice/', farmers.cropprice,name='cropprice'),

path('cropanalysies/', farmers.cropanalysies,name='cropanalysies'),

path('adminloginentered/',admn.adminloginentered,name='adminloginentered'),

path('adminlogin/',admn.adminlogin,name='adminlogin'),

path('knnalgo/',admn.knnalgo,name='knnalgo'),

path('rf1/',admn.rf1,name='rf1'),

path('lstmtest/',farmers.lstmtest,name='lstmtest'),

path('logout/',admn.logout,name='logout'),

if settings.DEBUG:

urlpatterns += static(settings.MEDIA\_URL,document\_root=settings.MEDIA\_ROOT)

**views.py:**

from django.shortcuts import render

from django.http import HttpResponse

from django.contrib import messages

from farmers.forms import \*

import csv

import io

def index(request):

return render(request,'index.html')

def farmerLogin(request):

return render(request,'farmer/farmerlogin.html')

def farmerpage(request):

return render(request,'farmer/farmerpage.html')

def farmerregister(request):

if request.method == 'POST':

form1=farmerModelform(request.POST)

if form1.is\_valid():

form1.save()

print("succesfully saved the data")

return render(request,"farmer/farmerlogin.html")

#return HttpResponse("registreration succesfully completed")

else:

print("form not valied")

return HttpResponse("form not valied")

else:

form=farmerModelform()

return render(request,"farmer/farmerregister.html",{"form":form})

def farmerlogincheck(request):

if request.method == 'POST':

sname = request.POST.get('email')

print(sname)

spasswd = request.POST.get('pswd')

print(spasswd)

try:

check = farmerModel.objects.get(email=sname, password=spasswd)

# print('usid',usid,'pswd',pswd)

print(check)

# request.session['name'] = check.name

# print("name",check.name)

status = check.status

print('status',status)

if status == "Activated":

request.session['email'] = check.email

return render(request, 'farmer/farmerpage.html')

else:

messages.success(request, 'farmer is not activated')

return render(request, 'farmer/farmerlogin.html')

except Exception as e:

print('Exception is ',str(e))

pass

messages.success(request,'Invalid name and password')

return render(request,'farmer/farmerlogin.html')

def farmerdetails(request):

s=farmerModel.objects.all()

return render(request,'farmer/farmerdetails.html',{"qs":s})

def activatefarmer(request):

if request.method =='GET':

uname=request.GET.get('pid')

print(uname)

status='Activated'

print("pid=",uname,"status=",status)

farmerModel.objects.filter(id=uname).update(status=status)

qs=farmerModel.objects.all()

return render(request,"farmer/farmerdetails.html",{"qs":qs})

def storecsvdata(request):

if request.method == 'POST':

# if request.method == "GET":

# return render(request, template, prompt)

csv\_file = request.FILES['file']

# let's check if it is a csv file

if not csv\_file.name.endswith('.csv'):

messages.error(request, 'THIS IS NOT A CSV FILE')

data\_set = csv\_file.read().decode('UTF-8')

# setup a stream which is when we loop through each line we are able to handle a data in a stream

io\_string = io.StringIO(data\_set)

next(io\_string)

for column in csv.reader(io\_string, delimiter=',', quotechar="|"):

\_, created = storecropdatamodel.objects.update\_or\_create(

state=column[0],

crop=column[1],

minprice=column[2],

maxprice=column[3],

year=column[4],

label=column[5],

)

context = {}

return render(request,'admn/storecsvdata.html')

def cropdata(request):

qs=storecropdatamodel.objects.all()

l1=[]

for x in qs:

l=x.crop

l1.append(l)

s=set(l1)

print("set-crop",s)

# qs1={"obj":s}

return render(request,'farmer/cropdata.html',{"obj":s})

def cropprice(request):

qs=storecropdatamodel.objects.all()

return render(request,'farmer/cropprice.html',{"obj":qs})

import numpy

import pandas as pd

from sklearn.model\_selection import GridSearchCV

from keras.wrappers.scikit\_learn import KerasClassifier

from keras.models import Sequential

from keras.layers import Dense

import warnings

warnings.filterwarnings('ignore')

def lstmtest(request):

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

from subprocess import check\_output

from keras.layers.core import Dense, Activation, Dropout

from keras.layers.recurrent import LSTM

from keras.models import Sequential

from sklearn.model\_selection import train\_test\_split

import time # helper libraries

from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt

from numpy import newaxis

prices\_dataset = pd.read\_csv('agr2.csv', header=0)

print(prices\_dataset.head())

yahoo = prices\_dataset[prices\_dataset['crop'] == 'tomato']

yahoo\_stock\_prices = yahoo.maxprice.values.astype('float32')

yahoo\_stock\_prices = yahoo\_stock\_prices.reshape(15, 1)

print(yahoo\_stock\_prices.shape)

plt.plot(yahoo\_stock\_prices)

plt.show()

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

yahoo\_stock\_prices = scaler.fit\_transform(yahoo\_stock\_prices)

train\_size = int(len(yahoo\_stock\_prices) \* 0.80)

test\_size = len(yahoo\_stock\_prices) - train\_size

train, test = yahoo\_stock\_prices[0:train\_size, :], yahoo\_stock\_prices[train\_size:len(yahoo\_stock\_prices), :]

print(len(train), len(test))

# convert an array of values into a dataset matrix

def create\_dataset(dataset, look\_back=1):

dataX, dataY = [], []

for i in range(len(dataset) - look\_back - 1):

a = dataset[i:(i + look\_back), 0]

dataX.append(a)

dataY.append(dataset[i + look\_back, 0])

return np.array(dataX), np.array(dataY)

look\_back = 1

trainX, trainY = create\_dataset(train, look\_back)

testX, testY = create\_dataset(test, look\_back)

trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

# Step 2 Build Model

model = Sequential()

model.add(LSTM(

input\_dim=1,

output\_dim=50,

return\_sequences=True))

model.add(Dropout(0.2))

model.add(LSTM(

100,

return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(

output\_dim=1))

model.add(Activation('linear'))

start = time.time()

model.compile(loss='mse', optimizer='rmsprop')

print('compilation time : ', time.time() - start)

model.fit(

trainX,

trainY,

batch\_size=128,

nb\_epoch=10,

validation\_split=0.05)

def plot\_results\_multiple(predicted\_data, true\_data, length):

plt.plot(scaler.inverse\_transform(true\_data.reshape(-1, 1))[length:])

plt.plot(scaler.inverse\_transform(np.array(predicted\_data).reshape(-1, 1))[length:])

plt.show()

# predict lenght consecutive values from a real one

def predict\_sequences\_multiple(model, firstValue, length):

prediction\_seqs = []

curr\_frame = firstValue

for i in range(length):

predicted = []

# print(model.predict(curr\_frame[newaxis,:,:]))

predicted.append(model.predict(curr\_frame[newaxis, :, :])[0, 0])

curr\_frame = curr\_frame[0:]

curr\_frame = np.insert(curr\_frame[0:], i + 1, predicted[-1], axis=0)

prediction\_seqs.append(predicted[-1])

return prediction\_seqs

predict\_length = 5

predictions = predict\_sequences\_multiple(model, testX[0], predict\_length)

# print("Predictions ",np.array(predictions))

price = scaler.inverse\_transform(np.array(predictions).reshape(-1, 1))

print("Predection price = ", price)

return HttpResponse("Under LSTM Constructions")

# print('predictions price is = ',predictions)

# plot\_results\_multiple(predictions, testY, predict\_length)

# plt.show()

# for index, value in enumerate(predictions):

# print(index, value)

return render(request,’’admn/lstm.html”)

**models.py:**

from django.db import models

class farmerModel(models.Model):

name = models.CharField(max\_length=100)

password = models.CharField(max\_length=100)

mobile = models.CharField(max\_length=100)

email = models.CharField(max\_length=100)

state = models.CharField(max\_length=100)

status = models.CharField(max\_length=100)

def \_\_str\_\_(self):

return self.email

class Meta:

db\_table='farmers'

class storecropdatamodel(models.Model):

state = models.CharField(max\_length=100)

crop = models.CharField(max\_length=100)

minprice = models.CharField(max\_length=100)

maxprice = models.CharField(max\_length=100)

year = models.CharField(max\_length=100)

label = models.CharField(max\_length=100)

def \_\_str\_\_(self):

return self.state, self.crop, self.minprice, self.maxprice, self.year, self.label

class Meta:

db\_table = 'storecropdata'

**cropdata.html:**

{% extends 'farmerbase.html'%}

{% load static %}

{% block contents %}

<section class="banner\_part">

<div class="container">

<div class="row align-items-center justify-content-center">

<div class="col-lg-7">

<div class="banner\_text text-center">

<div class="banner\_text\_iner">

<h4 style="color:#B7D044">

</h4>

<p><table border="2px solid red" align="left">

<tr><th style="color:green">Id</th>

<th style="color:green">crop-data</th>

</tr>

<tr>

<td style="color:red">{{ 1 }}</td>

<td style="color:red">{{ obj }}</td>

</tr>

</table>

</p>

<!--<a href="#" class="btn\_2"></a>-->

</div>

</div>

</div>

</div>

</div>

</section>

{% endblock %}