SOURCE CODE

APP.PY

import numpy as np import streamlit as st import pandas as pd import datetime

import plotly.graph\_objects as go import base64

import time import tensorflow

st.set\_page\_config( page\_title=" DEEP WIND ", page\_icon="►†"

)

old\_models =tensorflow.keras.models.load\_model('model.h5')

# set background, use base64 to read local file def get\_base64\_of\_bin\_file(bin\_file):

with open(bin\_file, 'rb') as f: data = f.read()

return base64.b64encode(data).decode()

def set\_png\_as\_page\_bg(png\_file):

bin\_str = get\_base64\_of\_bin\_file(png\_file) page\_bg\_img = '''

<style>

body {

background-image: url("data:image/png;base64,%s"); background-size: cover;

}

</style>

''' % bin\_str

st.markdown(page\_bg\_img, unsafe\_allow\_html=True) return

set\_png\_as\_page\_bg('gr.gif')

def home():

return "welcome"

def predict(temperature,pressure,wind\_speed,wind\_direction): values=np.array([[temperature,pressure,wind\_speed,wind\_direction]])

prediction=old\_models.predict(values.reshape(-1,1,4), batch\_size=1) print(prediction)

return prediction

def main():

st.sidebar.markdown("<h1 style='text-align: center; color: black;'> unsafe\_allow\_html=True)

Navigation Bar ◆g˙G×x•</h1>",

nav = st.sidebar.radio("",["Home ","User defined Prediction. —⬛","Forecasting ç#[¡⎛"])

if nav == "Home ":

st.markdown("<h1 style ='color:black; text\_align:center;font-family:times new roman;font- size:20pt; font-weight: bold;'>DEEP WINDS ”\(</h1>", unsafe\_allow\_html=True) st.markdown("<h1 style=' color:brown; text\_align:center;font-weight: bold;font-size:19pt;'>Made by Quad Techies with </h1>", unsafe\_allow\_html=True)

st.markdown("<h1 style ='color:black; text\_align:center;font-family:times new roman;font-weight: bold;font-size:16pt;'>◆ WIND POWER PREDICTION DL WEB-APP ◆ </h1>", unsafe\_allow\_html=True)

if nav == "User defined Prediction . —⬛":

set\_png\_as\_page\_bg('gra (1).jpg')

st.markdown("<h1 style='text-align: center; color: green;'>User Input Parameters □□</h1>", unsafe\_allow\_html=True)

with st.beta\_expander("Preferences"):

st.markdown("<h1 style='text-align: left; font-weight:bold;color:black;background-

color:white;font-size:11pt;'> Temperature •\* col1,col2 = st.beta\_columns(2)

with col1:

~½ (°C) </h1>",unsafe\_allow\_html=True)

min\_temp=st.number\_input(' )\· Minimum Temperature (°C)',min\_value=- 89,max\_value=55,value=-15,step=1)

with col2:

max\_temp=st.number\_input('· \) Maximum Temperature (°C)',min\_value=- 88,max\_value=56,value=50,step=1)

st.markdown("<h1 style='text-align: left; font-weight:bold;color:black;background- color:white;font-size:11pt;'> Wind Speed '˜F (m/s) </h1>",unsafe\_allow\_html=True)

col1,col2 = st.beta\_columns(2) with col1:

min\_speed=st.number\_input(' Minimum Wind Speed

(m/s)',min\_value=0,max\_value=99,value=1,step=1) with col2:

max\_speed=st.number\_input(' Maximum Wind Speed

(m/s)',min\_value=2,max\_value=100,value=27,step=1) st.write("")

temperature = st.slider('Temperature •\* max\_value=max\_temp,value=max\_temp)

~½ [°C]', min\_value=min\_temp, step=1,

pressure = st.slider('Pressure ) [atm]️', 0.9, 1.0, 1.0)

wind\_speed = st.slider('Wind Speed ˜ 'F [m/s]', min\_value=min\_speed, step=1,

max\_value=max\_speed,value=max\_speed)

wind\_direction = st.slider('Wind Direction ►†‡ [deg]', 0, 1, 360) result = ""

if st.button("Predict"):

result = predict(temperature,pressure,wind\_speed,wind\_direction) st.balloons()

st.success('Predicted Power is {} kW'.format(result))

if nav == "Forecasting [⎛¡#ç":

set\_png\_as\_page\_bg('04.gif')

st.markdown("<h1 style='text-align: center; color:black ;'>)FORECASTING)</h1>", unsafe\_allow\_html=True)

# Setup file upload

st.markdown("<h1 style='text-align:center; color:white;background-color:black;font-size:14pt'>)y Upload your CSV or Excel file. (200MB max) )y</h1>", unsafe\_allow\_html=True)

uploaded\_file = st.file\_uploader(label="",type=['csv', 'xlsx']) global df

if uploaded\_file is not None:

print(uploaded\_file)

st.markdown("<h1 style='text-align:center; color:black;background-color:lightgreen;font- size:14pt'>)y File upload successful )y</h1>", unsafe\_allow\_html=True)

try:

df = pd.read\_csv(uploaded\_file) st.write(df)

except Exception as e:

df = pd.read\_excel(uploaded\_file) st.write(df)

st.markdown("<h1 style='text-align: center; color:black ;background-color:powderblue;font- size:14pt'>⬛#/ INPUT DATA #/⬛</h1>", unsafe\_allow\_html=True)

trace = go.Scatter(x = df['DateTime'], y = df['Power generated by system | (kW)'],mode = 'lines',name = 'Data')

layout = go.Layout(title = "",xaxis = {'title' : "Date"},yaxis = {'title' : "Power generated by system | (kW)"})

fig = go.Figure(data=[trace], layout=layout) st.write(fig)

df1=df.reset\_index()['Power generated by system | (kW)'] import matplotlib.pyplot as plt

st.write("\n")

st.markdown("<h1 style='text-align: center; color:black ;background-color:powderblue;font- size:14pt'>/⬛# INPUT DATA IN TERMS OF NO. OF HOURS ⬛#/</h1>",

unsafe\_allow\_html=True)

trace = go.Scatter(x = df1.index,y = df['Power generated by system | (kW)'],mode = 'lines', name = 'Data' )

layout = go.Layout(title = "",xaxis = {'title' : "No. of hours"},yaxis = {'title' : "Power generated by system (kW)"})

fig = go.Figure(data=[trace], layout=layout) #fig.show()

st.write(fig)

from sklearn.preprocessing import MinMaxScaler scaler=MinMaxScaler(feature\_range=(0,1)) df1=scaler.fit\_transform(np.array(df1).reshape(-1,1))

##splitting dataset into train and test split training\_size=int(len(df1)\*0.65) test\_size=len(df1)-training\_size

train\_data,test\_data=df1[0:training\_size,:],df1[training\_size:len(df1),:1]

import numpy

# convert an array of values into a dataset matrix # convert an array of values into a dataset matrix def create\_dataset(dataset, time\_step=1):

dataX, dataY = [], []

for i in range(len(dataset)-time\_step-1):

a = dataset[i:(i+time\_step), 0] ###i=0, 0,1,2,3-----99 100 dataX.append(a)

dataY.append(dataset[i + time\_step, 0]) return numpy.array(dataX), numpy.array(dataY)

# reshape into X=t,t+1,t+2,t+3 and Y=t+4 time\_step = 30

X\_train, y\_train = create\_dataset(train\_data, time\_step) X\_test, ytest = create\_dataset(test\_data, time\_step)

# reshape input to be [samples, time steps, features] which is required for LSTM X\_train =X\_train.reshape(X\_train.shape[0],X\_train.shape[1] , 1)

X\_test = X\_test.reshape(X\_test.shape[0],X\_test.shape[1] , 1) # Create the BILSTM model

from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import LSTM

from tensorflow.keras.layers import Bidirectional model = Sequential()

model.add(Bidirectional(LSTM(250, input\_shape=(1, 30)))) model.add(Dense(1))

model.compile(loss='mae', optimizer='adam')

model.fit(X\_train,y\_train,validation\_data=(X\_test,ytest),epochs=10,batch\_size=64,verbose=1) import tensorflow as tf

train\_predict=model.predict(X\_train) test\_predict=model.predict(X\_test)

#Transformback to original form train\_predict=scaler.inverse\_transform(train\_predict) test\_predict=scaler.inverse\_transform(test\_predict)

### Calculate RMSE performance metrics import math

from sklearn.metrics import mean\_squared\_error math.sqrt(mean\_squared\_error(y\_train,train\_predict))

### Test Data RMSEmath.sqrt(mean\_squared\_error(ytest,test\_predict)) ### Plotting

# shift train predictions for plotting look\_back=30

trainPredictPlot = numpy.empty\_like(df1) trainPredictPlot[:, :] = np.nan

trainPredictPlot[look\_back:len(train\_predict)+look\_back, :] = train\_predict # shift test predictions for plotting

testPredictPlot = numpy.empty\_like(df1) testPredictPlot[:, :] = numpy.nan

testPredictPlot[len(train\_predict)+(look\_back\*2)+1:len(df1)-1, :] = test\_predict # plot baseline and predictions

st.markdown("<h1 style='text-align: center; color:black ;background-color:powderblue;font-

size:14pt'>#/⬛ TRAIN AND TEST DATA </h1>", unsafe\_allow\_html=True)

#plt.plot(scaler.inverse\_transform(df1)) plt.plot(scaler.inverse\_transform(df1), color="blue", linewidth=1, linestyle="-") plt.xlabel('No. of hours')

# Set the y axis label of the current axis. plt.ylabel('Power generated by system | (kW)')

plt.plot(trainPredictPlot,label='Train Data',color="black",linewidth=2, linestyle="--") plt.plot(testPredictPlot,label='Test Data',color="orange",linewidth=2, linestyle="--") plt.legend(loc="upper left")

#plt.show() st.pyplot(plt)

x\_input=test\_data[len(test\_data)-30:].reshape(1,-1) temp\_input=list(x\_input) temp\_input=temp\_input[0].tolist()

# demonstrate prediction for next 24 hours from numpy import array

lst\_output=[] n\_steps=30 i=0

while(i<24): if(len(temp\_input)>30): #print(temp\_input)

x\_input=np.array(temp\_input[1:]) x\_input=x\_input.reshape(1,-1)

x\_input = x\_input.reshape((1, n\_steps, 1)) yhat = model.predict(x\_input, verbose=0) temp\_input.extend(yhat[0].tolist()) temp\_input=temp\_input[1:] lst\_output.extend(yhat.tolist())

i=i+1

else:

x\_input = x\_input.reshape((1, n\_steps,1)) yhat = model.predict(x\_input, verbose=0)

print(yhat[0]) temp\_input.extend(yhat[0].tolist())

print(len(temp\_input)) lst\_output.extend(yhat.tolist()) i=i+1

print(lst\_output) day\_new=np.arange(1,31) day\_pred=np.arange(len(df1),len(df1)+24)

import matplotlib.pyplot as plt print(len(df1)) progress=st.progress(0)

for i in range(100): time.sleep(0.1) progress.progress(i+1) st.balloons()

st.markdown("<h1 style='text-align: center; color:black ;background-color:powderblue;font- size:14pt'>/#⬛ PREDICTED RESULTS FOR NEXT 24 HOURS /#⬛</h1>",

unsafe\_allow\_html=True) plt.plot(day\_pred,scaler.inverse\_transform(lst\_output),color="green",linewidth=1.5, linestyle="-- ",marker='\*',markerfacecolor='yellow', markersize=7)

plt.legend('GTTP',loc="upper left")

plt.xlabel('No. of hours')

# Set the y axis label of the current axis. plt.ylabel('Power generated by system | (kW)')

st.pyplot(plt)

st.markdown("<h1 style='text-align: center; color:black ;background-color:yellow;font- size:14pt'> z° G-Given Data, \n° °zT-Train Data, \n° °zT-Test Data, \n° °zP-Predicted Results</h1>", unsafe\_allow\_html=True)

st.write(scaler.inverse\_transform(lst\_output))

if name == " main ":

main()

MODEL .PY

import pandas as pd import datetime import numpy as np

from keras.models import Sequential from keras.layers import Dense

from keras.layers import LSTM

from keras.layers import Bidirectional import pandas as pd

import keras

''' Loading data '''

df = pd.read\_excel('Dataset.csv') df=df.drop(columns=['DateTime']) ''' Cleaning Data ''' #dataframe.drop['Date'].values

df['Power generated by system | (kW)'].replace(0, np.nan, inplace=True) df['Power generated by system | (kW)'].fillna(method='ffill', inplace=True)

X = df.drop(columns=['Power generated by system | (kW)']) Y = df[['Power generated by system | (kW)']] X=np.array(X).reshape(-1,1,4)

Y=np.array(Y).reshape(-1,1,1)

model = Sequential()

model.add(Bidirectional(LSTM(100, activation='relu',input\_shape=(-1,1,4)))) model.add(Dense(1))

model.compile(loss='mae', optimizer='adam',metrics=['accuracy'])

model.fit(X, Y,epochs=100,callbacks=[keras.callbacks.EarlyStopping(patience=3)])

test\_data = np.array([[-4.858,0.989741,6.651,273]]) o=model.predict(test\_data.reshape(-1,1,4), batch\_size=1) print(o)

# Saving model to disk models=model.save('model.h5')