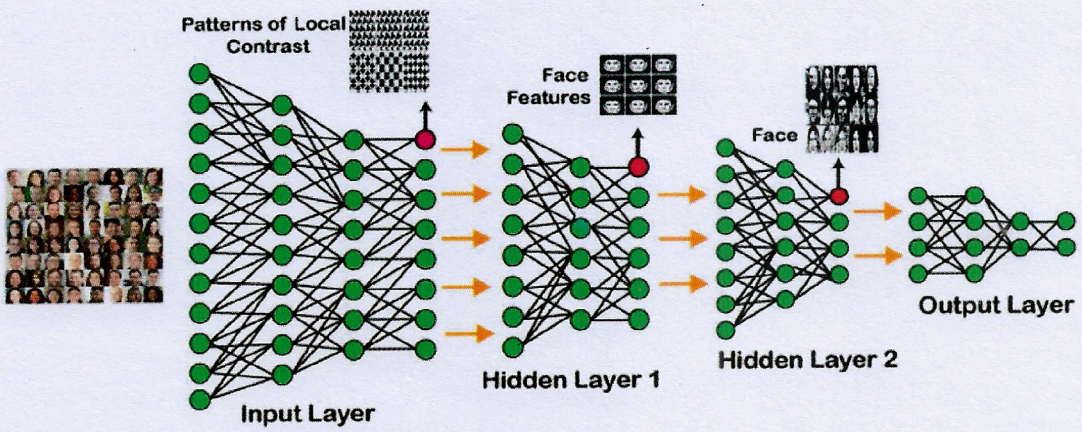
INTERNSHIP PROJECTS

1. WIND SPEED AND WIND DIRECTION PREDICTION USING

TECHNOLOGY USED:

- **Python**: Python offers a user-friendly syntax that makes it easy to learn and write code. Its extensive library ecosystem, including popular frameworks like TensorFlow provides vast capabilities for various domains, including web development, data analysis, machine learning, and automation.
- Python's versatility allows it to be used for small scripting tasks as well as large-scale applications. With its strong community support and active development, Python continues to evolve, making it an excellent choice for both beginners and experienced programmers alike.
- **Time Series Analysis** is a statistical technique used to analyze and model time-based data. It is used in various fields such as finance, economics, and engineering to analyze patterns and trends in data over time. The goal of time series analysis is to identify the underlying patterns, trends, and seasonality in the data, and to use this information to make informed predictions about future values
- **Deep learning** is based on the branch of machine learning, which is a subset of artificial intelligence. Since neural networks imitate the human brain and so deep learning will do. In deep learning, nothing is programmed explicitly. Basically, it is a machine learning class that makes use of numerous nonlinear processing units so as to perform feature extraction as well as transformation. The output from each preceding layer is taken as input by each one of the successive layers.
- Deep learning is implemented with the help of Neural Networks, and the idea behind the motivation of Neural Network is the biological neurons, which is nothing but a brain cell.
- Deep learning models are capable enough to focus on the accurate features themselves by requiring a little guidance from the programmer and are very helpful in solving out the problem of dimensionality. Deep learning algorithms are used, especially when we have a huge no of inputs and outputs.
- Since deep learning has been evolved by the machine learning, which itself is a subset of artificial intelligence and as the idea behind the artificial intelligence is to mimic the human behavior, so same is "the idea of deep learning to build such algorithm that can mimic the brain".

- Example of Deep Learning:



Packages imported here:

- `**import os**`: The `os` module provides functions for interacting with the operating system, allowing tasks such as file and directory manipulation, environment variables access, and executing system commands.

- `**import mysql.connector**`: The `mysql.connector` module functions for interacting with the Mysql databases, allowing tasks such as Selection from a table, Insertion into a table, and several SQL querys.

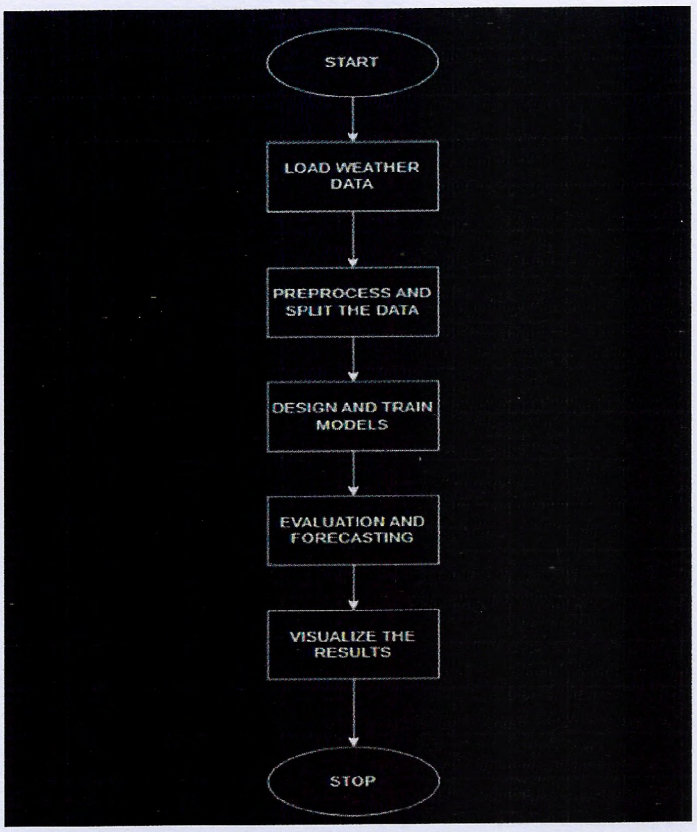
- `import time`: The `time` module provides functions for working with time-related operations, including measuring time intervals, pausing program execution, and obtaining the current time.

- `**import pandas as pd**`: The `pandas` library is a popular data manipulation and analysis tool in Python, providing data structures (e.g., DataFrames) and functions to handle and manipulate structured data efficiently.

- `import numpy as np`: The `numpy` library is a fundamental package for scientific computing in Python. It provides high-performance multidimensional array objects, along with a large collection of mathematical functions to operate on these arrays.

- `import matplotlib.pyplot as plt`: The `matplotlib.pyplot` module is a plotting library in Python, allowing the creation of various types of visualizations, such as line plots, bar charts, histograms, and scatter plots.

- `import tensorflow as tf`: TensorFlow is a popular open-source library for machine learning and deep learning. Importing it as `tf` allows access to a wide range of functions and classes for building and training neural networks.



- Start: The flowchart begins.
- Load Data: Load historical wind speed data from a dataset or external source.
- Preprocess and split Data: Perform data pre-processing steps such as handling missing values, removing outliers, and normalizing the data. Split the preprocessed data into training and testing datasets.
- Design and train Model: Select a suitable neural network architecture for wind speed prediction and forecasting. Initialize the neural network model with the chosen architecture and hyperparameters. Train the model using the training dataset. Adjust the model parameters using an optimization algorithm, such as Adam, and update the weights based on the model's performance.
- Evaluate Model and forecast: Evaluate the trained model's performance using the testing dataset. Calculate relevant metrics like mean squared error (MSE), mean absolute error (MAE), and R-squared. Use the trained model to make wind speed predictions for future time periods based on available input data.
- Visualize Results: Visualize the predicted wind speed values alongside the actual wind speed values to assess the model's accuracy and effectiveness.
- End: The flowchart ends.

PYTHON CODE: (Executed on Virtual Studio Code -VS Code)

❖ ConversionFrom1SecTo5Mins:

- This code is used to convert the 1 second data into 5 mins data and store the data in Data Base.

```
# -*- coding: utf-8 -*-
""" Created on Tue Aug 8 14:18:15 2023
@author: sandhya"""
import mysql.connector
config = {
'host': '127.0.0.1',
'port': 0000,
'database': 'wind',
'user': 'anish_mannem',
'password': 'root',
'charset': 'utf8',
'use_pure':True,
'use_unicode': True,
'get_warnings': True,}
conn=mysql.connector.connect(**config)
cursor=conn.cursor()
def convert_and_store_data():
# month=4
# year=2023
stmt_select_ws_wddir = "select ws.RECVDTTIME, ws.Level1,
wd.Level1 from WindSpeed_4_2023 as ws, winddir_4_2023 as wd
where ws.RECVDTTIME>1680287400494 AND
ws.RECVDTTIME<1682879399217 AND
ws.RECVDTTIME=wd.RECVDTTIME"
cursor.execute(stmt_select_ws_wddir)
windspeed_4_2023=cursor.fetchall()
# interval_minutes=5*60
interval_start=windspeed_4_2023[0][0]
```

```
interval_end=interval_start+(5*60*1000)
accumulate_value_ws=0
accumulate_value_wd=0
count=0
data_points_count=0
for timestamp,value1,value2 in windspeed_4_2023:
if(timestamp>=interval_end and data_points_count != 0):
date=int(interval_start/1000)
timestmp=date*1000
average_value_ws=accumulate_value_ws/data_points_count
average_value_wd=accumulate_value_wd/data_points_count
store_converted_data(timestmp,round(average_value_ws,3),round(ave
rage_value_wd,3))
# print(timestamp,round(average_value,3))
count=count+1
# cursor.execute("INSERT into
five_minute_data(timestarp,average_value)values(u,v)"),(interval_start
,average_value))
interval_start=interval_end
interval_end+=(5*60*1000)
value1=0
value2=0
accumulate_value_ws=0
accumulate_value_wd=0
data_points_count=0
else:
accumulate_value_ws+=value1
accumulate_value_wd+=value2
data_points_count+=1
print(count)
def store_converted_data(timestamp,average_value_ws,average_value_wd):
```

```
query = "insert into convert_2_5_min_data VALUES ('%d','%f','%f') "
% (timestamp,average_value_ws,average_value_wd)
cursor.execute(query)
conn.commit()
print(str(timestamp),average_value_ws,average_value_wd)
convert_and_store_data()
if len(data)==0:
print("no values found")
elif len(data)>=300:
last_120_data=data[-120:]
average=sum([record[1] for record in last_120_data])/120
print(f"average of last 120 records:{average}")
else:
average=sum([record[1] for record in data])/len(data)
print(f"average of available data:{average}")
```

❖ Prediction u:

This code is used for the Prediction of normalized Wind Speed which we have to determine in order to get the actual Wind Speed and store the data in the data base.

```
import mysql.connector
import numpy as np
import pandas as pd
import time
from datetime import datetime
import tensorflow as tf
import gc
RECVDTTIME = []
wsp = []
time1=[]
wdir = []
cnx = mysql.connector.connect(host="127.0.0.1",user="anish_mannem",password="root",database="wind")
```

```
cur = cnx.cursor()
# month = time.localtime().tm_mon;
# year=time.localtime().tm_year;
month = 4;
year=2023;
if month==1:
prevmonth = 12;
prevyear = year-1;
else:
prevmonth = month-1;
prevyear = year;
def connect_to_database():
cnx=mysql.connector.connect(host="127.0.0.1",user="anish_mannem",password="root",database="wind")
return cnx
new_model = tf.keras.models.load_model('test_model')
#def retrieve_existing_data(cnx):
# cur=cnx.cursor()
# cur.execute("SELECT * FROM converted_data ORDER BY timestamp_coloumn DESC LIMIT 72") #converted
data is 5 min data
#retrieve_existing_data("wind")
cur=cnx.cursor()
cur.execute("SELECT * FROM convert_2_5_min_data")
converted_data=cur.fetchall()
#converted data is 5 min data
#existing_data=cur.fetchall()
tim5=[]
wsp5=[]
dub_tim=[]
wdir5=[]
time3=[]
start=0
```

```
end=0
while(end<=len(converted_data)):
end=start+72
print(start,end)
for row in converted_data[start:end]:
tim5=np.append(tim5,datetime.utcfromtimestamp(int(row[0]/1000)))
wsp5=np.append(wsp5,float(row[1]))
wdir5=np.append(wdir5,float(row[2]))
df = pd.DataFrame()
df['datetime'] = tim5[start:end]
df['wd100'] = wdir5[start:end]
df['ws100'] = wsp5[start:end]
# df =df1[(df1['datetime'].dt.minute%5==0)&(df1['datetime'].dt.second==0)]
df['u'] = -(df['ws100']+0.000000001) * np.sin
(np.deg2rad(df['wd100']))
df['v'] = -(df['ws100']+0.000000001) * np.cos
(np.deg2rad(df['wd100']))
df['doy'] = df['datetime'].dt.dayofyear
df['minute'] = df['datetime'].dt.minute
df['hour'] = df['datetime'].dt.hour
df['week'] = df['datetime'].dt.week
df['month'] = df['datetime'].dt.month
df['year'] = df['datetime'].dt.year
df['season'] = np.where((df['datetime'].dt.month >= 1) &
(df['datetime'].dt.month <= 2), 1,
np.where((df['datetime'].dt.month >= 3) &
(df['datetime'].dt.month <= 5), 2,
np.where((df['datetime'].dt.month >= 6) &
(df['datetime'].dt.month <= 9), 3,
np.where((df['datetime'].dt.month >= 10) &
(df['datetime'].dt.month <= 12), 4, 0))))
```

```
start=start+72
features_considered = ['year', 'season', 'month', 'week', 'doy', 'hour',
'minute', 'u']
features = df[features_considered]
dataset = features.values
#normalize the data
# data_mean = dataset.mean(axis=0)
# data_std = dataset.std(axis=0)
data_std = np.array([ 1.11794145, 1.01708552, 3.43074182,
15.01433025,
105.09135916, 6.90985834, 17.26054416, 3.758597677])
data_mean=np.array([ 2.01648813e+03, 2.67480778e+00,
6.51318591e+00, 2.66283521e+01,
1.82918789e+02, 1.15147897e+01, 2.74997983e+01, -3.16725206e01])
dataset = (dataset-data_mean)/data_std
if(len(df)==72):
data = []
data.append(dataset[0:72])
output=[]
output.append([0,0,0,0,0,0,0,0,0,0,0,0])
# # test_data_multi = tf.data.Dataset.from_tensor_slices((data,output))
test_data_multi = test_data_multi.batch(1)
# # Load the saved model
# new_model = tf.keras.models.load_model('u_model.h5')
#Target Feature
target_feature = 'u'
target_idx = features.columns.get_loc(target_feature)
for x, y in test_data_multi.take(1):
ypred=np.array((new_model.predict(x)[0]*data_std[target_idx])+data_mean[target_idx])
month = time.localtime().tm_mon;
year=time.localtime().tm_year;
```

```
if float(df['datetime'].values[-1].astype(float)/1000000) in dub_tim:

print("duplicate value")

else:

dub_tim.append(float(df['datetime'].values[-1].astype(float)/1000000));

stmt_select = "INSERT INTO prediction_u1 VALUES ('%d','%f','%f','%f','%f','%f','%f','%f','%f','%f','%f','%f','%f');"
\% \ (float(df['datetime'].values[-
1].astype(float)/1000000),float(ypred[0]),float(ypred[1]),float(ypred[2]),float(ypred[3]),float(ypred[4]),float(
ypred[5]),float(ypred[6]),float(ypre

d[7]),float(ypred[8]),float(ypred[9]),float(ypred[10]),float(ypred[11]))

print(float(df['datetime'].values[-1].astype(float)/1000000))

cur.execute(stmt_select)

cnx.commit()

print("Prediction Success:: ",time.localtime(),gc.collect())

else:

print("Record Length Not matching::",time.localtime(), len(df))

del df['datetime']

del df['ws100']

del df['year']

del df['wd100']

del df['season']

del df['month']

del df['week']

del df['doy']

del df['hour']

del df['minute']

del df['u']

del df['v']

cnx.close()

mysql.connector.connect.close()
```

Prediction v:

- This code is used for the Prediction of normalized Wind Speed which we have to determine in order to get the actual Wind Speed and store the data in the data base.

```
import mysql.connector
import numpy as np
import pandas as pd
import time
from datetime import datetime
import tensorflow as tf
import gc
RECVDTTIME = []
wsp = []
wdir = []
cnx =
mysql.connector.connect(host="127.0.0.1",user="anish_mannem",password="root",database="wind")
cur = cnx.cursor()
# month = time.localtime().tm_mon;
# year=time.localtime().tm_year;
month = 4;
year=2023;
if month==1:
prevmonth = 12;
prevyear = year-1;
else:
prevmonth = month-1;
prevyear = year;
def connect_to_database():
cnx=mysql.connector.connect(host="127.0.0.1",user="anish_mannem",password="
root",database="wind")
return cnx
#connect_to_database()
```

```
# Load the saved model
new_model = tf.keras.models.load_model('test_model')
#prediction
#def retrieve_existing_data(cnx):
# cur=cnx.cursor()
# cur.execute("SELECT * FROM converted_data ORDER BY
timestamp_coloumn DESC LIMIT 72") #converted data is 5 min data
# existing_data=cur.fetchall()
# return existing_data
#retrieve_existing_data("wind")
cur=cnx.cursor()
cur.execute("SELECT * FROM convert_2_5_min_data")
converted_data=cur.fetchall()
#converted data is 5 min data
#existing_data=cur.fetchall()
tim5=[]
wsp5=[]
wdir5=[]
dub_tim=[]
start=0
end=0
while(end<=len(converted_data)):
end=start+72
print(start,end)
for row in converted_data[start:end]:
tim5=np.append(tim5,datetime.utcfromtimestamp(int(row[0]/1000)))
wsp5=np.append(wsp5,float(row[1]))
wdir5=np.append(wdir5,float(row[2]))
df = pd.DataFrame()
df['datetime'] = tim5[start:end]
df['wd100'] = wdir5[start:end]
```

```
df['ws100'] = wsp5[start:end]
# df =df1[(df1['datetime'].dt.minute%5==0)&(df1['datetime'].dt.second==0)]
df['u'] = -(df['ws100']+0.000000001) * np.sin
(np.deg2rad(df['wd100']))
df['v'] = -(df['ws100']+0.000000001) * np.cos
(np.deg2rad(df['wd100']))
df['doy'] = df['datetime'].dt.dayofyear
df['minute'] = df['datetime'].dt.minute
df['hour'] = df['datetime'].dt.hour
df['week'] = df['datetime'].dt.week
df['month'] = df['datetime'].dt.month
df['year'] = df['datetime'].dt.year
df['season'] = np.where((df['datetime'].dt.month >= 1) &
(df['datetime'].dt.month <= 2), 1,
np.where((df['datetime'].dt.month >= 3) &
(df['datetime'].dt.month <= 5), 2,
np.where((df['datetime'].dt.month >= 6) &
(df['datetime'].dt.month <= 9), 3,
np.where((df['datetime'].dt.month >= 10) &
(df['datetime'].dt.month <= 12), 4, 0))))
start=start+1
features_considered = ['year', 'season', 'month', 'week', 'doy', 'hour', 'minute', 'v']
features = df[features_considered]
dataset = features.values
#normalize the data
# data_mean = dataset.mean(axis=0)
# data_std = dataset.std(axis=0)
data_std = np.array([ 1.11794145, 1.01708552, 3.43074182, 15.01433025,105.09135916, 6.90985834,
17.26054416, 3.758597677])
data_mean=np.array([ 2.01648813e+03, 2.67480778e+00,
6.51318591e+00, 2.66283521e+01,
```

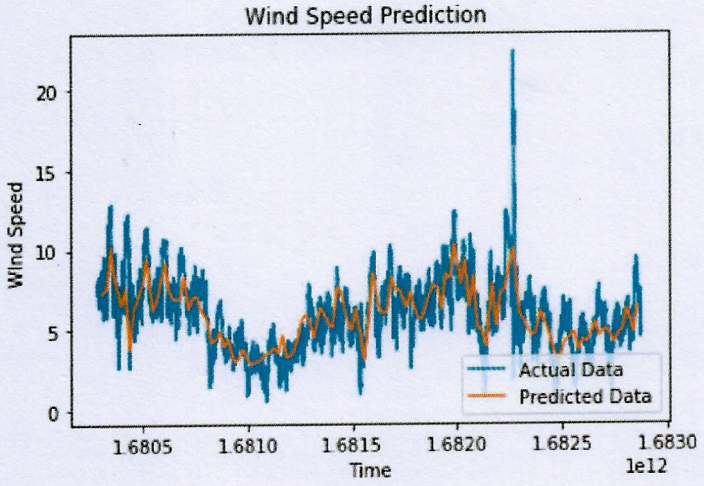
```
1.82918789e+02, 1.15147897e+01, 2.74997983e+01, -3.16725206e01])

dataset = (dataset-data_mean)/data_std

if(len(df)==72):

data = []

data.append(dataset[0:72])

output=[]

output.append([0,0,0,0,0,0,0,0,0,0,0,0])

# #test_data_multi = tf.data.Dataset.from_tensor_slices((data,output))

test_data_multi = test_data_multi.batch(1)

# # Load the saved model

# new_model = tf.keras.models.load_model('u_model.h5')

#Target Feature

target_feature = 'v'

target_idx = features.columns.get_loc(target_feature)

for x, y in test_data_multi.take(1):

ypred=np.array((new_model.predict(x)[0]*data_std[target_idx])+data_mean[target_idx])

# print(df)

# print(ypred)

# Save predicted Wind direction

month = time.localtime().tm_mon;

year=time.localtime().tm_year;

if float(df['datetime'].values[-1].astype(float)/1000000) in dub_tim:

print("duplicate value")

else:

dub_tim.append(float(df['datetime'].values[-1].astype(float)/1000000));

stmt_select = "INSERT INTO prediction_v1 VALUES ('%d','%f','%f','%f','%f','%f','%f','%f','%f','%f','%f','%f','%f');"
\% \(float(df['datetime'].values[-
1].astype(float)/1000000),float(ypred[0]),float(ypred[1]),float(ypred[2]),float(ypred[3]),float(ypred[4]),float(
ypred[5]),float(ypred[6]),float(ypred[7]),float(ypred[8]),float(ypred[9]),float(ypred[10]),float(ypred[11]))

print(float(df['datetime'].values[-1].astype(float)/1000000))

cur.execute(stmt_select)

cnx.commit()
```

```
print("Prediction Success:: ",time.localtime(),gc.collect())
else:
print("Record Length Not matching::",time.localtime(), len(df))
del df['datetime']
del df['ws100']
del df['year']
del df['wd100']
del df['season']
del df['month']
del df['week']
del df['doy']
del df['hour']
del df['minute']
del df['u']
del df['v']
cnx.close()
mysql.connector.connect.close()
```

❖ Graph_Wind speed:

- This code is used to compare the actual and predicted Wind Speed

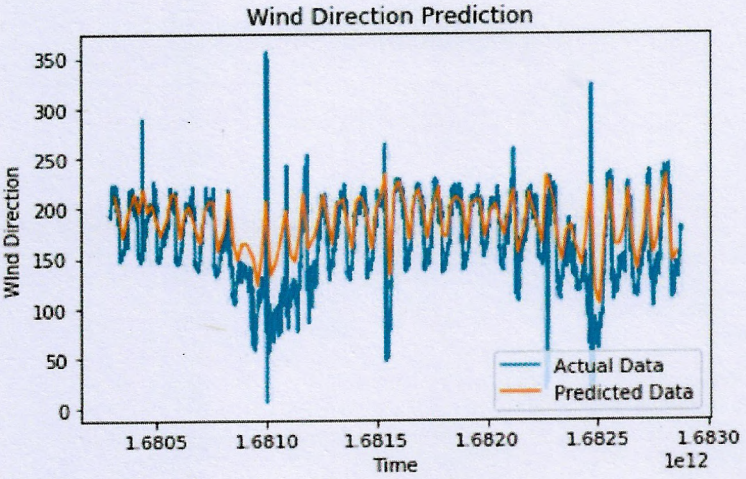
```
import matplotlib.pyplot as plt
import pandas as pd
import mysql.connector
config = {
'host': '127.0.0.1',
'port': 0000,
'database': 'wind',
'user': anish_mannem,
'password': 'root',
'charset': 'utf8',
'use_pure':True,
'use_unicode': True,
'get_warnings': True,}
conn=mysql.connector.connect(**config)
cursor=conn.cursor()
query="SELECT RCVDTTIME as time,wsp from
convert_2_5_min_data"
data=pd.read_sql(query,conn)
plt.plot(data['time'],data['wsp'],label='a_wspeed')
query="SELECT u.RECVDTTIME as time,u.p_wsp as wsp from
prediction_u1 as u"
data=pd.read_sql(query,conn)
plt.plot(data['time'],data['wsp'],label='p_wspeed')
conn.close()
plt.title("Wind Speed Prediction ")
plt.xlabel("Time")
plt.ylabel("WInd Speed")
plt.legend(['Actual Data', 'Predicted Data'], loc=4)
plt.time('graph')
```



❖ Graph_Wind direction:

- This code is used to compare the actual and predicted Wind Direction

```
import matplotlib.pyplot as plt
import pandas as pd
import mysql.connector
config = {
'host': '127.0.0.1',
'port': 0000,
'database': 'wind',
'user': anish_mannem,
'password': 'root',
'charset': 'utf8',
'use_pure':True,
'use_unicode': True,
'get_warnings': True,}
conn=mysql.connector.connect(**config)
cursor=conn.cursor()
query="SELECT RCVDTTIME as time,wsp from
convert_2_5_min_data"
data=pd.read_sql(query,conn)
plt.plot(data['time'],data['wsp'],label='a_wdirection')
query="SELECT v.RECVDTTIME as time,v.p_wsp as wsp from
prediction_v1 as v"
data=pd.read_sql(query,conn)
plt.plot(data['time'],data['wsp'],label='p_wdirection')
conn.close()
plt.title("Wind Direction Prediction ")
plt.xlabel("Time")
plt.ylabel("WInd Direction")
plt.legend(['Actual Data', 'Predicted Data'], loc=4)
plt.time('graph')
```



ANALYSIS:

In conclusion, this research is aimed to Predict the wind speed and Wind Direction. By analyzing 6 hours of historical data, we successfully predicted wind speeds and directions for 1 hours into the future. Additionally, we addressed the challenge of missing data by implementing the grouper function of the panda's library to group time intervals into 5-minute intervals and filling the null values accordingly.

The results of our study demonstrated the effectiveness of the selected models in accurately forecasting wind speeds. The inclusion of the additional 1-hour forecast window allowed for more comprehensive and timely predictions, which can greatly benefit various industries such as renewable energy, transportation, and emergency management.

Furthermore, the utilization of the grouper function from the panda's library provided an efficient solution to handle missing data. By grouping time intervals into smaller increments, we were able to capture the underlying patterns and trends in the data, leading to more reliable predictions. This methodology can be applied to other time-series forecasting tasks where missing data is a common challenge.

Overall, this research contributes to the field of wind speed and direction forecasting by presenting a practical approach for predicting future wind speeds based on historical data. The findings highlight the importance of accurate wind speed predictions in various applications and emphasize the significance of addressing missing data in time-series analysis. Future studies can build upon these results by exploring additional models and refining the data preprocessing techniques to further enhance the accuracy and reliability of wind speed and direction forecasting.