GRT INSTITUTE OF ENGINEERING AND TECHNOLOGY, TIRUTTANI - 631209

Approved by AICTE, New Delhi Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROJECT TITLE

Air quality analysis and prediction in tamilnadu

COLLEGE CODE:1103

Revathi.M

3rd yr, 5th sem

Reg no:110321104040

revathi0112m@gmail.com

EXPLANATION:

Analyzing and predicting air quality in Tamil Nadu, like in any other region, involves a combination of data collection, monitoring, and modeling. Here's a detailed explanation of the process:

Data Collection:

Monitoring Stations: Establish a network of air quality monitoring stations across Tamil Nadu. These stations measure various air pollutants, including particulate matter (PM2.5, PM10), nitrogen dioxide (NO2), sulfur dioxide (SO2), carbon monoxide (CO), ozone (O3), and volatile organic compounds (VOCs).

Meteorological Data:

Gather meteorological data such as temperature, humidity, wind speed, and wind direction, which can influence air quality.

Data Analysis:

Historical Data: Analyze historical air quality data to identify trends and patterns. This can help in understanding seasonal variations and the impact of pollution sources.

Correlation Analysis:

Determine relationships between meteorological conditions and air quality. For example, how wind patterns affect the dispersion of pollutants.

Air Quality Index (AQI):

Calculate the AQI based on the concentration of different pollutants. The AQI provides a simple way to communicate air quality to the public.

Source Identification:

Identify major sources of pollution in Tamil Nadu, such as industries, vehicular emissions, construction activities, and agricultural practices.

Modeling:

Develop air quality models that use historical data, meteorological data, and source information to predict future air quality. Common models include the Community Multiscale Air Quality (CMAQ) model and the Weather Research and Forecasting with Chemistry (WRF-Chem) model.

Machine Learning:

Utilize machine learning algorithms to improve prediction accuracy. Machine learning can help in understanding complex patterns in air quality data and making real-time predictions.

Public Awareness:

Communicate air quality information to the public through websites, mobile apps, and public announcements. Provide health advisories based on the AQI to protect vulnerable populations.

Policy and Mitigation:

Use the information to formulate and enforce air quality regulations. Implement measures to reduce emissions from major pollution sources.

Validation:

Continuously validate the accuracy of predictions by comparing them to real-time monitoring data. Adjust models and predictions as needed.

Emergency Response:

Develop contingency plans for extreme pollution events, such as smog or wildfires, to protect public health.

Research and Development:

Invest in research to improve monitoring technology, modeling techniques, and pollution control measures.

Collaboration:

Collaborate with neighboring states and countries, as air quality is not confined by political boundaries.

The success of air quality analysis and prediction in Tamil Nadu relies on the integration of data, technology, and proactive policies to reduce pollution and protect public health. It's an ongoing process that requires continuous monitoring and adaptation to changing conditions and emission sources.

Dataset explanation:

A typical air quality dataset includes parameters like:

1. Particulate Matter (PM2.5 and PM10):
   Fine particles in the air that can have adverse health effects.

2. Ozone (O3):
   Ground-level ozone, which can cause respiratory problems.

3. Nitrogen Dioxide (NO2) and Sulfur Dioxide (SO2):

Gases produced by combustion processes.

4. Carbon Monoxide (CO):
   A colorless, odorless gas that can be harmful when inhaled.

5. Weather Data:
   Temperature, humidity, wind speed, and direction, which influence air quality.

6. Geographic Coordinates:
   Information about monitoring station locations.

Air quality prediction typically involves using machine learning models to forecast future air quality levels based on historical data. This can help in air quality management and alert systems.

Begin building the project by loading the dataset:

The columns involved in the dataset are

- Stn_code
- Sampling_data
- State
- Location
- Agency
- Type
- So2
- No2
- Rspm
- Spm
- Location_monitoring_station
- Pm2_5
- Date

To begin building the air quality analysis and prediction in tamilnadu project using the dataset from Kaggle, you'll need to load the dataset and start exploring it. You can use Python and popular libraries like Pandas for data manipulation and Matplotlib or Seaborn for data visualization. Here's a step-by-step guide:

Importing necessary libraries:

```
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

loading the dataset:

```
data1=pd.read_csv('C:/project/data.csv',encoding="ISO-8859-1")
data=data1.sample(500)
print(data)
```

Output:

```
stn_code sampling_date  ... pm2_5      date
128268     NaN     9/5/2009  ...   NaN  2009-05-09
119673      35   19/01/2012  ...   NaN  2012-01-19
433064     162     24-12-14  ...   NaN  2014-12-24
220657   508.0     13-12-10  ...   NaN  2010-12-13
35036    538.0    2/10/2012  ...   NaN  2012-10-02
...        ...          ...  ...   ...      ...
302214     NaN    21/2/2009  ...   NaN  2009-02-21
356914   766.0     17-05-13  ...   NaN  2013-05-17
326928     NaN     18-05-07  ...   NaN  2007-05-18
428065       9   13/07/2011  ...   NaN  2011-07-13
274319     NaN     30-08-07  ...   NaN  2007-08-30
[500 rows x 13 columns]
```

preprocessing the data:

1. Data Collection:

First, collect air quality data for Tamil Nadu from reliable sources like government agencies or research organizations.

2. Data Cleaning:

- Remove duplicates, missing values, and outliers.

- Convert data types, if needed.

```
# Remove duplicates
data= data.drop_duplicates()
print(data)
#Handle missing values
data= data.dropna()
print(data)

# Handle outliers (you can define your own outlier detection method)
from scipy import stats

z_scores = stats.zscore(data['PM2.5'])
```

```
data = data[(z_scores < 3)]
print(data)
```

3.Data Transformation:

- Convert date/time columns to datetime objects.

- Extract features like year, month, day, and hour for time series analysis.

```
# Convert 'Date' column to datetime format
data['Date'] = pd.to_datetime(data['Date'])
print(data)
```

4. Feature Engineering:

- Create new features or modify existing ones if they can improve model performance.

```
#future engineering
data['Year'] = data['Date'].dt.year
data['Month'] = data['Date'].dt.month
```

5. Normalization/Scaling:

- Normalize or scale features, especially if you're using algorithms sensitive to feature scales.

6.Outliers detection and handling:

- Identify and handle outliers in numerical columns if necessary. You can use methods like Z-score, IQR, or domain-specific rules.

```
# Step 6
: Outlier Detection and Handling (example: using Z-score)
from scipy import stats
z_scores = stats.zscore(data[['SO2', 'NO2', 'RSPM', 'SPM', 'PM2_5']])
data = data[(z_scores < 3).all(axis=1)]  # Remove outliers
```

7. Splitting the Dataset:

- Split the dataset into training, validation, and test sets for model evaluation.

```
X = data.drop('Target_Column', axis=1)  # Replace 'Target_Column' with your
target variable
y = data['Target_Column']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

8. Save Preprocessed Data:

- Save the preprocessed data to a new file for future analysis to avoid repeating these steps.

```
# Step 8: Save Preprocessed Data
data.to_csv('preprocessed_data.csv', index=False)
```

Performing data analysis :

```
data.fillna(0, inplace=True)
data.head()
#Function to calculate so2 individual pollutant index(si)
def calculate_si(so2):
    si=0
    if (so2<=40):
     si= so2*(50/40)
    if (so2>40 and so2<=80):
     si= 50+(so2-40)*(50/40)
    if (so2>80 and so2<=380):
     si= 100+(so2-80)*(100/300)
    if (so2>380 and so2<=800):
     si= 200+(so2-380)*(100/800)
    if (so2>800 and so2<=1600):
     si= 300+(so2-800)*(100/800)
    if (so2>1600):
     si= 400+(so2-1600)*(100/800)
    return si
data['si']=data['so2'].apply(calculate_si)
df= data[['so2','si']]
df.head()
def calculate_ni(no2):
    ni=0
    if(no2<=40):
     ni= no2*50/40
    elif(no2>40 and no2<=80):
     ni= 50+(no2-14)*(50/40)
    elif(no2>80 and no2<=180):
     ni= 100+(no2-80)*(100/100)
    elif(no2>180 and no2<=280):
     ni= 200+(no2-180)*(100/100)
    elif(no2>280 and no2<=400):
     ni= 300+(no2-280)*(100/120)
    else:
     ni= 400+(no2-400)*(100/120)
    return ni
data['ni']=data['no2'].apply(calculate_ni)
df= data[['no2','ni']]
df.head()
def calculate_(rspm):
    rpi=0
    if(rpi<=30):
     rpi=rpi*50/30
    elif(rpi>30 and rpi<=60):
     rpi=50+(rpi-30)*50/30
    elif(rpi>60 and rpi<=90):
     rpi=100+(rpi-60)*100/30
    elif(rpi>90 and rpi<=120):
     rpi=200+(rpi-90)*100/30
    elif(rpi>120 and rpi<=250):
     rpi=300+(rpi-120)*(100/130)
```

```
    else:
     rpi=400+(rpi-250)*(100/130)
    return rpi
data['rpi']=data['rspm'].apply(calculate_si)
df= data[['rspm','rpi']]
df.tail()
#many data values of rspm values is unawailable since it was not measure
before
#Function to calculate no2 individual pollutant index(spi)
def calculate_spi(spm):
    spi=0
    if(spm<=50):
     spi=spm
    if(spm<50 and spm<=100):
     spi=spm
    elif(spm>100 and spm<=250):
     spi= 100+(spm-100)*(100/150)
    elif(spm>250 and spm<=350):
     spi=200+(spm-250)
    elif(spm>350 and spm<=450):
     spi=300+(spm-350)*(100/80)
    else:
     spi=400+(spm-430)*(100/80)
    return spi
data['spi']=data['spm'].apply(calculate_spi)
df= data[['spm','spi']]
df.tail()
#many data values of rspm values is unawailable since it was not measure
before
#function to calculate the air quality index (AQI) of every data value
#its is calculated as per indian govt standards
def calculate_aqi(si,ni,spi,rpi):
    aqi=0
    if(si>ni and si>spi and si>rpi):
     aqi=si
    if(spi>si and spi>ni and spi>rpi):
     aqi=spi
    if(ni>si and ni>spi and ni>rpi):
     aqi=ni
    if(rpi>si and rpi>ni and rpi>spi):
     aqi=rpi
    return aqi
data['AQI']=data.apply(lambda
x:calculate_aqi(x['si'],x['ni'],x['spi'],x['rpi']),axis=1)
df= data[['sampling_date','state','si','ni','rpi','spi','AQI']]
df.head()
df.state.unique()
state=pd.read_csv("../input/indian-states-lat-lon/lat.csv")
state.head()
df.head()
dff=pd.merge(state.set_index("state"),df.set_index("state"),
right_index=True, left_index=True).reset_index()
dff.head()
from mpl_toolkits.basemap import Basemap
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

```
%config InlineBackend.figure_format = 'retina'
m = Basemap(projection='mill',llcrnrlat=5,urcrnrlat=40,
llcrnrlon=60,urcrnrlon=110,lat_ts=20,resolution='c')
longitudes = dff["lon"].tolist()
latitudes = dff["lat"].tolist()
#m = Basemap(width=12000000,height=9000000,projection='lcc',
            #resolution=None,lat_1=80.,lat_2=55,lat_0=80,lon_0=-107.)
x,y = m(longitudes,latitudes)
fig = plt.figure(figsize=(12,10))
plt.title("All affected areas")
m.plot(x, y, "o", markersize = 3, color = 'blue')
m.drawcoastlines()
m.fillcontinents(color='coral',lake_color='aqua')
m.drawmapboundary()
m.drawcountries()
plt.show()
#Visualization of AQI across india

data['date'] = pd.to_datetime(data['date'],format='%Y-%m-%d') # date parse
data['year'] = data['date'].dt.year # year
data['year'] = data['year'].fillna(0.0).astype(int)
data = data[(data['year']>0)]

df =
data[['AQI','year','state']].groupby(["year"]).median().reset_index().sort_va
lues(by='year',ascending=False)
f,ax=plt.subplots(figsize=(15,10))
sns.pointplot(x='year', y='AQI', data=df)
#setting up date parameter
import warnings
import itertools
import dateutil
import statsmodels.api as sm
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
%matplotlib inline
df=data[['AQI','date']]
df["date"] = pd.to_datetime(df['date'])
df.tail(20)
#Calculating the yearly mean for the data
df=df.set_index('date').resample('M')["AQI"].mean()
df.head()
#preprocessing the data values
data=df.reset_index(level=0, inplace=False)
data = data[np.isfinite(data['AQI'])]
data=data[data.date != '1970-01-31']
data = data.reset_index(drop=True)
data.head()
#visualizing the processed data of AQI
df=data.set_index('date')
df.sort_values(by='date',ascending=False)
df.plot(figsize=(15, 6))
plt.show()
y=df.AQI
#exctracting knowledge about data
#spliting dataframes into test and train
```

```
n = df.shape[0]
train_size = 0.65
features_dataframe = df.sort_values('date')
train = df.iloc[:int(n * train_size)]
test = df.iloc[int(n * train_size):]
#plotting the yearly variations of AQI
train.AQI.plot(figsize=(15,8), title= 'YEARLY VARIATIONS', fontsize=14)
test.AQI.plot(figsize=(15,8), title= 'YEARLY VARIATIONS', fontsize=14)
plt.show()
#Naive Forecast Approach to find the variations(trend)

dd= np.asarray(train.AQI)
y_hat = test.copy()
y_hat['naive'] = dd[len(dd)-1]
plt.figure(figsize=(12,8))
plt.plot(train.index, train['AQI'], label='Train')
plt.plot(test.index,test['AQI'], label='Test')
plt.plot(y_hat.index,y_hat['naive'], label='Naive Forecast')
plt.legend(loc='best')
plt.title("Naive Forecast",fontsize=20)

plt.legend(["actual ","predicted"])
plt.xlabel("YEAR",fontsize=20)
plt.ylabel("AQI",fontsize=20)
plt.tick_params(labelsize=20)
plt.show()
#various statmodel to identity huge variations od data values
import statsmodels.api as sm
train.index=pd.DatetimeIndex(freq="w", start=0 ,periods=224)

sm.tsa.seasonal_decompose(train.AQI).plot()
result = sm.tsa.stattools.adfuller(train.AQI)
plt.show()
#resampling the data to predict monthly AQI of india
df=data[['AQI','date']]
df['date']=pd.to_datetime(df['date'])
date=df.groupby(pd.Grouper(key='date',freq='1MS'))["AQI"].mean()
df.count()
#splitting the sampling date into month and year accordingly
data['month'] = data['date'].dt.month
data['year'] = data['date'].dt.year
data=data[['AQI','date','month','year']]
data.head()
#Appling BOXPLOT analysis
df =
data[['AQI','year']].groupby(["year"]).mean().reset_index().sort_values(by='y
ear',ascending=False)
df=df.dropna()
dd=df
df.describe()
import seaborn as sns
sns.boxplot(x=df['AQI'])
#removing Outliers
df = df[np.isfinite(df['AQI'])]
df=df[df.AQI >153]
df=df[df.AQI <221]
#visualizing the filttered data
```

```
year=df['year'].values
AQI=df['AQI'].values
df['AQI']=pd.to_numeric(df['AQI'],errors='coerce')
df['year']=pd.to_numeric(df['year'],errors='coerce')

import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (20.0, 10.0)
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(year,AQI, color='red')
plt.show()
#scatter plot of data points
cols =['year']
y = df['AQI']
x=df[cols]

plt.scatter(x,y)
plt.show()
# Predicted val
newB = [200.17, -1.54]


def rmse(y, y_pred):
    rmse = np.sqrt(sum(y - y_pred))
    return rmse


y_pred = x.dot(newB)

dt = pd.DataFrame({'Actual': y, 'Predicted': y_pred})
x = pd.concat([df, dt], axis=1)
x
x
#calculating the root mean squared error for the predicted AQi values
from sklearn import metrics
print(np.sqrt(metrics.mean_squared_error(y,y_pred)))
x_axis=x.year
y_axis=x.Actual
y1_axis=x.Predicted
plt.plot(x_axis,y_axis)
plt.plot(x_axis,y1_axis)
plt.title("Actual vs Predicted",fontsize=20)
plt.legend(["actual ","predicted"])
plt.xlabel("YEAR",fontsize=20)
plt.ylabel("AQI",fontsize=20)
plt.tick_params(labelsize=20)
plt.show()
#applying boxplot analysis
import seaborn as sns
sns.boxplot(x=df['AQI'])
#plotting data points
cols =['year']
y = df['AQI']
x=df[cols]
```

```
plt.scatter(x,y)
plt.show()
#testing the accuracy of the model

from sklearn import metrics
print(np.sqrt(metrics.mean_squared_error(y,y_pred)))
#plotting the actual and predicted results
x_axis=x.year
y_axis=x.Actual
y1_axis=x.Predicted
plt.plot(x_axis,y_axis)
plt.plot(x_axis,y1_axis)
plt.title("Actual vs Predicted",fontsize=20)
plt.legend(["actual ","predicted"])
plt.xlabel("YEAR",fontsize=20)
plt.ylabel("AQI",fontsize=20)
plt.tick_params(labelsize=20)
plt.show()
#Prediction for the future
from sklearn.preprocessing import MinMaxScaler
#feeding in the x value-years
data=[[-1,2016],[-1,2017],[-1,2018],[-1,2019],[-1,2020]]
#normalization
scaler=MinMaxScaler(feature_range=(-1,1))
scaler.fit(data)
x=scaler.transform(data)
#calculations
newB=[103.59,-2.74]
ypred=-(x.dot(newB))
#AQI for the year 2020
print("AQI for the year 2020===>",ypred[-1])
```