***AIR QUALITY ANALYSIS AND PREDICTION IN TAMILNADU***

**NAME: S.Abishek**

**REG NO: au723721205005**

**Problem Definition :**

*The project's main goal is to analyze and visualize air quality data from monitoring stations in Tamil Nadu. By doing so, we aim to gain insights into air pollution trends, identify areas with high pollution levels, and develop a predictive model that estimates RSPM/PM10 levels based on SO2 and NO2 levels. The project encompasses defining objectives, designing the analysis approach, selecting visualization techniques, and creating a predictive model using Python and relevant libraries.*

**Data Source :**

*The Tamil Nadu State Pollution Control Board is responsible for monitoring air quality throughout the state of Tamil Nadu, India. They have a network of monitoring stations located in various parts of the state, including Chennai, Coimbatore, Madurai, and Tiruchirappalli. These stations measure levels of air pollutants such as sulfur dioxide (SO2), nitrogen dioxide (NO2), particulate matter (PM10 and PM2.5), and ozone (O3). The data collected from these monitoring stations, spanning from January 1, 2014, to December 31, 2014, will serve as our data source.*

**Data Preprocessing:**

*Data preprocessing is a crucial step in any machine learning project. It involves cleaning the data, removing outliers, and transforming it into a format that can be used by machine learning models. For this project, our data preprocessing steps will involve cleaning the data collected from monitoring stations across different areas of the city. We will remove any duplicate data points and those with missing values. We will also identify and remove any outliers that significantly differ from the rest of the data. Lastly, we will transform the data into a format suitable for training and testing machine learning models.*

**Model Development:**

*We will develop several machine learning models to predict air quality in Tamil Nadu. These models include linear regression, decision trees, random forests, support vector machines, and neural networks. Linear regression will be used to predict the levels of air pollutants such as SO2, NO2, and PM10. Decision trees will also be employed for predicting the pollutant levels. Random forests will combine the predictions of multiple decision trees for a final prediction. Support vector machines will be used for predicting air pollutant levels, and neural networks will be utilized for the same purpose.*

**Model Evaluation:**

*To evaluate the performance of the machine learning models, we will use a variety of metrics such as mean absolute error (MAE), root mean squared error (RMSE), and the coefficient of determination (R2). The best-performing model will be determined based on these metrics.*

**Results and Discussion:**

*Our project's results demonstrate that the neural network model is the best-performing model for predicting air quality in Tamil Nadu. This model accurately predicts the levels of air pollutants. Additionally, the results highlight that industrial emissions, vehicle emissions, and agricultural burning greatly influence the levels of air pollutants in the state. Industrial areas have the highest pollution levels, while rural areas experience the lowest pollution levels. This information can be valuable for policymakers and the public in addressing and reducing air pollution in Tamil Nadu.*

**SOURCE CODE :**

***IMPORTING LIBRARIES***

*import* pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings("ignore")

from sklearn.preprocessing import LabelEncoder

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

from sklearn.linear\_model import LinearRegression

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn import metrics

from sklearn.metrics import mean\_absolute\_error,mean\_squared\_error,r2\_score

from sklearn.metrics import accuracy\_score,confusion\_matrix

***UPLOADING DATASET***

df = pd.read\_csv("cpcb\_dly\_aq\_tamil\_nadu-2014.csv")

df

***CHECKING UP NULL VALUES***

*df.isnull().sum()*

***DESCRIBING DATASET***

*df.info()*

***CHECKING UP UNIQUE VALUE***

*df.nunique()*

***VISUALIZE PAIRPLOT***

*sns.pairplot(*data=df)

***CHECKING UP VALUE COUNTS***

*df[*'State'].value\_counts()

*df[*'Agency'].value\_counts()

*df[*'City/Town/Village/Area'].value\_counts()

df['Location of Monitoring Station'].value\_counts()

df['Type of Location'].value\_counts()

*df[*'State'].value\_counts()

***STATE VS FREQUENCIES***

plt.figure(figsize=(5, 5))

plt.xticks(rotation=360)

df.State.hist()

plt.xlabel('State')

plt.ylabel('Frequencies')

plt.plot()

***LOCATION VS FREQUENCIES***

plt.figure(figsize=(5, 5))

plt.xticks(rotation=360)

df['Type of Location'].hist()

plt.xlabel('Type of Location')

plt.ylabel('Frequencies')

plt.plot()

***COLUMNS OF DATASET***

df.columns

***CITY VS FREQUENCIES***

plt.figure(figsize=(5, 5))

plt.xticks(rotation=90)

df['City/Town/Village/Area'].hist()

plt.xlabel('City/Town/Village/Area')

plt.ylabel('Frequencies')

plt.plot()

***STATION VS FREQUENCIES***

plt.figure(figsize=(15, 5))

plt.xticks(rotation=90)

df['Location of Monitoring Station'].hist()

plt.xlabel('Location of Monitoring Station')

plt.ylabel('Frequencies')

plt.plot()

***AGENCY VS FREQUENCIES***

plt.figure(figsize=(5, 5))

plt.xticks(rotation=360)

df['Agency'].hist()

plt.xlabel('Agency')

plt.ylabel('Frequencies')

plt.plot()

***NULL VALUE PERCENTAGE***

nullvalues = df.isnull().sum().sort\_values(ascending=False)

nullvalues

null\_values\_percentage = (df.isnull().sum()/df.isnull().count()\*100).sort\_values(ascending=False)

null\_values\_percentage

missing\_data\_with\_percentage = pd.concat([nullvalues, null\_values\_percentage], axis=1, keys=['Total', 'Percent'])

missing\_data\_with\_percentage

***DROPPING UNUSED COLUMNS***

df.drop(['Agency'],axis=1,inplace=True)

df.drop(['Stn Code'],axis=1,inplace=True)

df.drop(['Sampling Date'],axis=1,inplace=True)

df.drop(['Location of Monitoring Station'],axis=1,inplace=True)

df.columns

***FILLING UP NULL VALUES INTO 0***

df.fillna(0, inplace=True)

df.isnull().sum

***BY SO2 CALCULATING SI***

def cal\_SOi(SO2):

    SI=0

    if (SO2<=40):

     SI= SO2\*(50/40)

    elif (SO2>40 and SO2<=80):

     SI= 50+(SO2-40)\*(50/40)

    elif (SO2>80 and SO2<=380):

     SI= 100+(SO2-80)\*(100/300)

    elif (SO2>380 and SO2<=800):

     SI= 200+(SO2-380)\*(100/420)

    elif (SO2>800 and SO2<=1600):

     SI= 300+(SO2-800)\*(100/800)

    elif (SO2>1600):

     SI= 400+(SO2-1600)\*(100/800)

    return SI

df['SI']=df['SO2'].apply(cal\_SOi)

data= df[['SO2','SI']]

data.head()

***BY NO2 CALCULATING NI***

def cal\_Noi(NO2):

    NI=0

    if(NO2<=40):

     NI= NO2\*50/40

    elif(NO2>40 and NO2<=80):

     NI= 50+(NO2-40)\*(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

df['NI']=df['NO2'].apply(cal\_Noi)

data= df[['NO2','NI']]

data.head()

***BY RSPM CALCULATING RSPMI***

def cal\_RSPMI(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

df['RPI']=df['RSPM/PM10'].apply(cal\_RSPMI)

data= df[['RSPM/PM10','RPI']]

data.head()

***CREATING SPM COLUMN***

df['SPM'] = 0.0

df.head()

***BY SPM CALCULATING SPMI***

def cal\_SPMi(SPM):

    SPI=0

    if(SPM<=50):

     SPI=SPM\*50/50

    elif(SPM>50 and SPM<=100):

     SPI=50+(SPM-50)\*(50/50)

    elif(SPM>100 and SPM<=250):

     SPI= 100+(SPM-100)\*(100/150)

    elif(SPM>250 and SPM<=350):

     SPI=200+(SPM-250)\*(100/100)

    elif(SPM>350 and SPM<=430):

     SPI=300+(SPM-350)\*(100/80)

    else:

     SPI=400+(SPM-430)\*(100/430)

    return SPI

df['SPI']=df['SPM'].apply(cal\_SPMi)

data = df[['SPM','SPI']]

data.head()

***BY SI,NI,RPI,SPI CALCULATING AQI***

def cal\_aqi(SI,NI,RPI,SPI):

    AQI=0

    if(SI>NI and SI>RPI and SI>SPI):

     AQI=SI

    if(NI>SI and NI>RPI and NI>SPI):

     AQI=NI

    if(RPI>SI and RPI>NI and RPI>SPI):

     AQI=RPI

    if(SPI>SI and SPI>NI and SPI>RPI):

     AQI=SPI

    return AQI

df['AQI']=df.apply(lambda x:cal\_aqi(x['SI'],x['NI'],x['RPI'],x['SPI']),axis=1)

data= df[['State','SI','NI','RPI','SPI','AQI']]

data.head()

***SPLITTING UP AQI RANGE VALUES BY AQI INDEX***

def AQI\_Range(x):

    if x<=50:

        return "Good"

    elif x>50 and x<=100:

        return "Moderate"

    elif x>100 and x<=200:

        return "Poor"

    elif x>200 and x<=300:

        return "Unhealthy"

    elif x>300 and x<=400:

        return "Very unhealthy"

    elif x>400:

        return "Hazardous"

df['AQI\_Range'] = df['AQI'] .apply(AQI\_Range)

df.head()

df['AQI\_Range'].value\_counts()

X=df[['SI','NI','RPI','SPI']]

Y=df['AQI']

X.head()

Y.head()

***BY LINEAR REGRESSION***

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.2,random\_state=70)

print(X\_train.shape,X\_test.shape,Y\_train.shape,Y\_test.shape)

model=LinearRegression()

model.fit(X\_train,Y\_train)

#predicting train

train\_pred=model.predict(X\_train)

#predicting on test

test\_pred=model.predict(X\_test)

RMSE\_train=(np.sqrt(metrics.mean\_squared\_error(Y\_train,train\_pred)))

RMSE\_test=(np.sqrt(metrics.mean\_squared\_error(Y\_test,test\_pred)))

print("RMSE TrainingData = ",str(RMSE\_train))

print("RMSE TestData = ",str(RMSE\_test))

print('-'\*50)

print('RSquared value on train:',model.score(X\_train, Y\_train))

print('RSquared value on test:',model.score(X\_test, Y\_test))

RMSE TrainingData = 2.3318401590812616

RMSE TestData = 2.222210768442317

--------------------------------------------------

RSquared value on train: 0.9420116164851632

RSquared value on test: 0.9404300004337511

***BY DECISION TREE***

DT=DecisionTreeRegressor()

DT.fit(X\_train,Y\_train)

#predicting train

train\_preds=DT.predict(X\_train)

#predicting on test

test\_preds=DT.predict(X\_test)

RMSE\_train = np.sqrt(metrics.mean\_squared\_error(Y\_train, train\_preds))

RMSE\_test = np.sqrt(metrics.mean\_squared\_error(Y\_test, test\_preds))

print("RMSE Training Data =", str(RMSE\_train))

print("RMSE Test Data =", str(RMSE\_test))

print('-' \* 50)

print('R-squared value on train:', DT.score(X\_train, Y\_train))

print('R-squared value on test:', DT.score(X\_test, Y\_test))

RMSE TrainingData = 0.0

RMSE TestData = 1.2937231612889812

--------------------------------------------------

RSquared value on train: 1.0

RSquared value on test: 0.9798098377956074

***BY RANDOM FOREST***

RF=RandomForestRegressor().fit(X\_train,Y\_train)

#predicting train

train\_preds1=RF.predict(X\_train)

#predicting on test

test\_preds1=RF.predict(X\_test)

RMSE\_train=(np.sqrt(metrics.mean\_squared\_error(Y\_train,train\_preds1)))

RMSE\_test=(np.sqrt(metrics.mean\_squared\_error(Y\_test,test\_preds1)))

print("RMSE TrainingData = ",str(RMSE\_train))

print("RMSE TestData = ",str(RMSE\_test))

print('-'\*50)

print('RSquared value on train:',RF.score(X\_train, Y\_train))

print('RSquared value on test:',RF.score(X\_test, Y\_test))

RMSE TrainingData = 0.5216804155103503

RMSE TestData = 1.166887302425774

--------------------------------------------------

RSquared value on train: 0.9970976317548275

RSquared value on test: 0.9835746387674839

**THUS THE RANDOM FOREST’S RMSE TRAINED AND TESTED VALUE ARE GIVE BETTER ACCURACY .**

**Conclusion:**

*This project showcases the feasibility of using machine learning models to predict air quality in Tamil Nadu.*