

**COMPARATIVE ANALYSIS OF DEEP LEARNING AND
STATISTICAL MODEL FOR AIR POLLUTANTAS
PREDICTION IN URBAN AREAS**

A Project Report Submitted in Partial fulfilment of the
requirements for the award of the degree of

MASTER OF TECHNOLOGY

IN

DATA SCIENCE

Submitted by

CH.Sai Gopala Krishna Deekshit

VP22CSEN0200011

Under the esteemed guidance of

Dr.Pragnyaban Mishra

Professor, Gitam.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GITAM

(Deemed to be University)

VISAKHAPATNAM

April-2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM INSTITUTE OF TECHNOLOGY

GITAM

(Deemed to be University)



DECLARATION

I, hereby declare that the project report entitled “**Comparative Analysis of Deep Learning and Statistical Models for Air Pollutants Prediction in Urban Area**” is an original work done in the Department of Computer Science and Engineering, GITAM Institute of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of MTech. in Data Science. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date:

Registration No.
VP22CSEN0200011

Name
CH.Sai Gopala Krishna Deekshit

Signature

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM INSTITUTE OF TECHNOLOGY
GITAM
(Deemed to be University)



CERTIFICATE

This is to certify that the project report entitled “**Comparative Analysis of Deep Learning and Statistical Models for Air Pollutants Prediction in Urban Areas**” is a Bonafide record of work carried out by **CH. Sai Gopala Krishna Deekshit, VP22CSEN0200011** students submitted in partial fulfilment of requirement for the award of degree of Master of Technology in Data Science.

Project Guide

Head of the Department

Dr.Pragnyaban Mishra

Dr.G. Lakshmeeswari

Professor

Associate Professor

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to the following individuals and institutions for their invaluable support and guidance throughout the completion of this project:

- Dr. Pragnyaban Mishra, Professor at Gandhi Institute of Technology and Management (Deemed to be University), for his exceptional guidance and unwavering support. His expertise and insights have played a pivotal role in shaping the outcome of this documentation.
- Dr. G. Lakshmeeswari, Associate Professor and Head of the Department of Computer Science and Engineering at Gandhi Institute of Technology and Management (Deemed to be University), for her continuous encouragement and visionary leadership. Her support has been a driving force behind my success in this project.
- Dr. K. Prasada Rao, Assistant Professor and Coordinator at Gandhi Institute of Technology and Management (Deemed to be University), for his meticulous coordination and assistance throughout the project. His efforts have greatly contributed to the smooth execution of the project.
- GITAM Deemed to be University, for providing the necessary resources and facilities essential for the effective implementation of this project.
- The Department of Computer Science and Engineering, specializing in Data Science, for providing me with a conducive environment to pursue my academic and research interests.

I extend my heartfelt thanks to everyone involved for their unwavering support, guidance, and encouragement throughout this journey.

Sincerely,

CH. Sai Gopala Krishna Deekshit

ABSTRACT

Fast city and factory expansion causes air pollution and poor quality. It is called a "silent public health emergency" since it harms people and the environment. So, humans to tackle this worldwide issue, air pollution estimates must be accurate. Deep learning-based forecasting systems can make air quality forecasts more accurately and efficiently than before. We examined how deep learning-based single-step forecasting models like LSTM, GRU, and a statistical model predicted five types of air pollution: NO₂, O₃, SO₂, and PM_{2.5} and PM₁₀. We utilized a public dataset from a Belfast air quality monitoring location for the empirical evaluation. It monitors air pollution. RMSE, MAE, and R² are used to evaluate predictive models. Deep learning models always exhibited the lowest RMSE compared to traditional models. Also, the deep learning model achieved the highest R² score.

Keywords: Air quality, machine learning, deep learning, predictive models, statistical methods

TABLE OF CONTENTS

Acknowledgement	i
Abstract	ii
List of Figures and Tables	iv
1. Introduction	1-3
1.1 Objective	1
1.2 Problem Statement	2-3
2. Literature Review	4-6
3. Feasibility Study	7-8
4. System Analysis	9-11
4.1 The Existing System	9
4.2 The Proposed System	9-11
5. System Design	12-19
6. Implementation	20-28
7. Overview Of Technologies	29-36
8. System Testing	37-39
9. Frontend Screens	40-43
10. Conclusion & Future Scope	44
11. References	45-46

List of Figures and Tables

Fig.No	Figure Name	Page No
5.1	System Architecture	12
5.2	Data Flow Diagram	13
5.3	Use case Diagram	15
5.4	Class Diagram	16
5.5	Activity Diagram	17
5.6	Sequence Diagram	18
5.7	Collaboration Diagram	18
5.8	Component Diagram	19
5.9	Deployment Diagram	19
6.1	Importing Libraries and Reading Dataset	21
6.2	Splitting the data into Training and Testing	22
6.3	Performing Statistical Tests	22
6.4	Selecting Features	23
6.5	Preprocessing	23
6.6	Designing The Network (LSTM)	24
6.7	Designing	24
6.8	Performance measures Calculations (LSTM)	25
6.9	Plotting Graphs (LSTM)	25
6.10	Designing the Network (CNN)	26
6.11	Training and Testing (CNN)	26
6.12	Plotting Graphs (CNN)	27
6.13	Combining LSTM and CNN	27
6.14	Training and Testing (LSTM+CNN)	27
6.15	Plotting Graphs (LSTM+CNN)	28
6.16	Performance Measures (LSTM+CNN)	28
8.1	Static Testing	41
8.2	Structural Testing	41
8.3	Behavioral Diagram	42
9.1	Dash Board	43
9.2	Login Page	43
9.2	Login Page	44
9.4	User Input	44
9.5	Charts	45
9.6	Comparison	45
9.7	User Input	45
9.8	User Input	46
9.9	Output Page	46

1.INTRODUCTION

Air pollution has grown into a big problem around the world in the last few years. There is a direct link between air pollution and health and well-being, as well as the environment. People have seen that breathing in dirty air increases the risk of death and illness from things like lung diseases, dementia, heart disease, and cancer. The air quality is getting worse because of many things, including industry, industrial emissions, traffic emissions (on land, in the air, and at sea), dust, and coal use. To teach about this problem and make people more aware of it, we need to work with professionals and other interested parties from different fields. As part of their duties, local governments have set up a lot of air quality tracking stations across the country to keep an eye on how much pollution is in the air. The information gathered from these kinds of tracking sites can be used to guess what pollution will be found. To stop air pollution and find places that need help with air pollution and its effects, it's important to be able to predict the quality of the air. But figuring out how to correctly model air quality is hard in and of itself, and it rests on the data that is provided and the modelling methods that are used. In the past few years, deep learning-based forecasts models have shown promise as a better and more efficient way to predict air quality than other methods.

1.1 OBJECTIVE

- Combine meteorological elements like temperature, wind speed, and wind direction with an air quality component. This function uses the pollutant's previous hour's concentration and its projected future hour's concentration. We divide the datetime index into hour, day, and month components for added characteristics that improve prediction accuracy and reduce mistakes.
- We wish to research how to forecast the five primary air contaminants and compare statistics and deep learning base models.
- We discuss the architecture and parameters for deep learning and statistical models that forecast pollution. This may be used in smart cities or to improve model accuracy.

1.2 PROBLEM STATEMENT

Predicting air quality is very important for lowering pollution and finding places that need help to lessen its affects. But it's hard to make accurate models of air quality because they depend on the modeling methods and data that are available.

1.3 SOFTWARE REQUIREMENTS

Software requirements list the computer resources and system requirements needed for a program to run well. These needs are seldom supplied with program downloads. Install these individually before loading the program.

Platform - Software runs on computer platforms. Hardware or software may form the framework. Platforms often involve computer design, operating systems, programming languages, and runtime tools.

When you talk about system needs (software), one of the first things that comes up is the operating system. Different models of the same line of operating systems may not be able to work with the same software, but past compatibility is usually kept. Most Windows XP applications won't operate on Windows 98, but occasionally it may. Similar to how software created with the latest Linux Kernel v2.6 features doesn't operate or compile on v2.2 or v2.4.

APIs and drivers— Specialized software that uses multiple hardware devices, such as advanced display ports, needs custom drivers or APIs. DirectX, a collection of Microsoft APIs primarily used for multimedia and game development, is an example of this.

Web browser—The basic browser that comes with a computer is used by most web apps and software that relies on the Internet. Users often choose Microsoft Internet Explorer to run on Microsoft Windows. This software uses ActiveX features, which are known to be vulnerable.

1.4 HARDWARE REQUIREMENTS

Every OS and software need hardware, or computer tools. Hardware requirements lists frequently include a hardware compatibility list (HCL). In particular, operating systems.

Hardware included in an HCL has been tested with a certain operating system or software. Following articles discuss hardware requirements.

Architecture –Every computer operating system is made to work with a certain type of computer architecture. Most software programs can only run on certain operating systems and computer platforms. While some operating systems and programs are architecture-independent, most need to be recompiled for a new architecture. Here is a list of popular operating systems and the platforms they are compatible with.

Processing power—It is important for any software that the central processing unit (CPU) has enough power. In most programs that use the x86 design, processing power is shown by the CPU type and clock speed. Many people overlook factors such as bus speed, cache, and MIPS when considering the speed and power of a CPU. Despite having the same clock speed, Athlon and Pentium CPUs can have varying levels of performance. Intel Pentium CPUs are frequently discussed in this context due to their widespread popularity.

Memory– Random access memory is responsible for storing all software when a computer is turned on. It is important to take into account the memory requirements of the application, operating system, supporting software, files, and ongoing processes in order to ensure optimal performance of multiple applications running simultaneously on a multitasking machine.

Secondary storage– The amount of hard drive space needed for a software package is determined by factors such as its size, the number of temporary files generated during installation or use, and the use of swap space if there is insufficient RAM.

Display adapter— Graphics tools and high-end games that demand a better computer graphics display typically require high-end display adapters.

Peripherals – Some software applications require extra power or functionality to utilize particular tools in a unique manner. Accessories include laptops, CD-ROM players, pointing devices, and network gadgets.

2. LITERATURE REVIEW

2.1 Short-term effects of air pollutants on daily mortality in the Stockholm County—A spatiotemporal analysis:

<https://www.sciencedirect.com/science/article/pii/S0013935120307490>

ABSTRACT: Short-term air pollution exposure is connected to daily mortality in several studies. Most of the data originates from major city research, thus little is known about how urban and rural impacts differ. We investigated whether air pollution cause daily cause-specific fatalities in Stockholm county and outside the metropolitan region in the short run. We estimated Sweden's daily ozone (O₃), nitrogen dioxide (NO₂), and fine and inhalable particulate matter (PM_{2.5} and PM₁₀) levels from 2005 to 2016 using a spatiotemporal random forest model. The model covered 1 km. We linked daily mortality rates for each local region with air pollution and temperature using Stockholm county market statistics (SAMS). A case-crossover study examined the short-term relationship between the four pollutants and non-accidental, cardiovascular, and lung deaths. We compared groups in and around Stockholm using SAMS population density and the county's 26 towns. The full-year study showed that most air contaminants had no influence on cause-specific mortality. During warmer months (April-September), non-accidental mortality increased by 4.58% (95% CI: 0.89% to 8.41%) and 2.21% (95% CI: 0.71% to 3.73%) for every 10 µg/m³ increase in lag 0-1 PM_{2.5} and O₃. Stockholm and SAMS, with more individuals, had greater associations. Comparing the 26 municipalities' short-term air pollution linkages revealed no major variations. The research revealed some evidence that air pollution is hazardous even in non-cities, but not enough to declare for sure. We consider this research a test run to determine how everyday air pollution affects mortality rates throughout Sweden.

2.2 Digital twin of atmospheric environment: Sensory data fusion for high-resolution PM_{2.5} estimation and action policies recommendation:

<https://ieeexplore.ieee.org/document/10015739>

ABSTRACT: Particulate matter under 2.5 microns is a major health hazard. Using ground monitors to estimate its concentration is unwise for many reasons. This work creates a digital twin (DT) of an air environment using observation and remote sensing

data. DT relies on feedback to alter future input data. Combining Random Forest and Gradient Boosting to estimate PM_{2.5} levels suggests approaches to reduce clumping. A basic optimization issue helps us recommend activities like spreading the cloud, activating air filters, and sending SMS notifications. The proposed DT pipeline's PM_{2.5} estimate has an RMSE of 38.94 and an R² of 0.728 (95% CI: 0.717–0.740). We also investigate quantitative ways for measuring each independent variable.

2.3 Identification of high impact factors of air quality on a national scale using big data and machine learning techniques:

https://www.researchgate.net/publication/336730179_Identification_of_High_Impact_Factors_of_Air_Quality_on_A_National_Scale_Using_Big_Data_and_Machine_Learning_Techniques

ABSTRACT: Understanding air quality factors helps regulate and stop air pollution. Air pollution has been linked to various issues in many studies. However, our current tools don't handle multicollinearity adequately or explain why crucial things matter. The majority of writing has focused on a city or small region and one perspective. Few studies examine a country's priorities or many factors. This research offers a countrywide multivariate examination of air quality's most critical elements. It will address research gaps. For as many essential parameters as feasible, 171 characteristics were collected and evaluated. They comprised ecological, social, economic, weather, and energy variables. In a "big data" scenario like this, non-linear machine learning approach Extreme Gradient Boosting (XGBoost) describes the connection and determines how essential each variable is. Geographical Information System (GIS) prepares factors and displays results. Bayesian Optimization fine-tunes XGBoost's parameters after comparing its performance to other models. An experiment in the U.S. shows that our approach can uncover air quality parameters. Six factors have the greatest impact on air quality. The six perspectives provide air pollution management and prevention suggestions.

2.4 Environmental and health impacts of air pollution: A review:

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7044178/>

ABSTRACT: Air pollution is one of our biggest challenges since it impacts the climate and raises sickness and mortality rates. Multiple forms of pollutants cause illness. Particulate Matter (PM), tiny particles that enter the respiratory system by inhaling,

causes lung, heart, reproductive, and central nervous system illnesses, and even cancer. Ozone defends against UV rays in the atmosphere, but too much at ground level may harm the heart and lungs. Nitrogen oxide, sulfur dioxide, dioxins, PAHs, and volatile organic compounds are harmful air pollutants. Carbon monoxide may kill in large doses. Depending on exposure, heavy metals like lead may harm or make you unwell for a long time. COPD, asthma, and bronchiolitis are the main lung disorders caused by the aforementioned substances. They may also cause lung cancer, heart illness, CNS issues, and skin ailments. Finally, pollution-induced climate change and natural disasters alter the distribution of many illnesses. We need to raise awareness and bring together scientists from diverse domains to address this problem. National and international organizations must discuss this threat's origins and find long-term solutions.

2.5 Comparative analysis of machine learning techniques for predicting air quality in smart cities:

<https://ieeexplore.ieee.org/document/8746201>

ABSTRACT: Air pollution is a major environmental issue for smart cities. City authorities can assess traffic and make decisions by monitoring pollution data in real time. Using internet-connected monitoring has revolutionized air quality prediction. Different machine learning algorithms have been used to forecast pollution, but they need to be compared to see how long they take to analyze different data sets. In this study, we employed four sophisticated regression approaches to predict pollution and compared them to determine the optimal data size and processing time model for air quality prediction. Apache Spark was used to evaluate and estimate pollution using many datasets. MAE and RMSE are used to compare regression models. The time it takes to execute each approach utilizing standalone learning and fitting Apache Spark hyperparameter settings was determined to obtain the model with the lowest time and error rate.

3. FEASIBILITY STUDY

Feasibility Study

A viability study checks to see if a plan or idea is possible. Viability studies look at a possible business or project clearly and objectively to find out its resources, long-term success, risks and opportunities, and strengths and flaws. When deciding if something is possible, you should think about how much it will cost and how much it will be worth.

Types Of Feasibility Study

A viability study checks to see how likely it is that the project will work out. In light of this, it is very important that possible funders and banks see the study as objective. In five different ways, you can do a feasibility study. Each way looks at a different area.

1. Technical Feasibility

This review is mostly about the band's tech gear. It lets companies know if their team and technology can turn ideas into solutions that work. The technology, software, and other basic needs of the suggested system must be looked at to see if it is possible. For example, a company wouldn't put Star Trek's transporters in their building because that's not doable.

2. Economic Feasibility

A project cost-benefit study is often part of this evaluation. Before spending, this helps people figure out if a project is possible, how much it will cost, and what its benefits are. It gives a fair evaluation of the project and boosts its trustworthiness, which helps leaders figure out how the plan will improve the company's bottom line.

3. Legal Feasibility

This study checks to see if the project breaks any rules about data security, licensing, or social media. Let's say a business wants to build an office building somewhere. A profitability study might show that the company's main site isn't right for that business. That company saved time and effort by admitting from the start that their idea couldn't work.

4. Operational Feasibility

For this evaluation, study needs to be done to see if and how well the project meets the organization's goals. Operational feasibility studies check how well a project plan meets the needs that were found during the requirements analysis part of system development.

5. Scheduling Feasibility

This review is the most important part of the project for its success. The job will fail if it's not finished on time. A company estimates the project's time potential to figure out how long it will take to finish.

Once all of these things have been looked at, the profitability study helps find any issues the project may have.

Within the project, there are limitations in terms of technology, budget, resources, and other issues.

There are limitations inside the company, such as money, marketing, exports, and so on.

There are also limitations from outside the company, such as the environment, laws and rules, transportation, and so on.

4. SYSTEM ANALYSIS

4.1 EXISTING SYSTEM:

They created a non-linear approach using Extreme Gradient Boosting (XGBoost) and Geographic Information approach (GIS) to examine national air quality parameters. To determine the most relevant air quality determinants, scientists collected and examined 171 features, including social, economic, meteorological, and environmental data. XGBoost, a non-linear machine learning algorithm, described the relationship and determined variable importance. GIS prepared the variables and assessed their impact. Their approach framework also compares XGBoost's classification performance to other machine learning models to demonstrate decision reasonableness. A US case study proved their structure worked. Experiments demonstrate that six things mainly affect air quality. These six elements are utilized to create realistic air pollution management and prevention plans.

4.1.1 DISADVANTAGES OF EXISTING SYSTEM:

1. The current work looks at important factors using the XGBoost method, which might not work well with time series data.
2. The current work doesn't look at how different methods compare to each other. Because of this absence, people might not fully understand the pros and cons of different methods.
3. The current work rates its success based on how well it sorts things into groups, which may not directly relate to how well it predicts things will go.

4.2 Proposed System:

We evaluated deep learning-based single-step forecasting models in this research. They predicted nitrogen dioxide (NO₂), ozone (O₃), sulphur dioxide (SO₂), and particulate matter (PM_{2.5} and PM₁₀) using auto regressive integrated moving average (ARIMA). Our empirical evaluation employed a public dataset from N Ireland's central Belfast air quality monitoring station. It measures air pollution. RMSE, MAE, and R² are used to evaluate predictive models.

4.2.1 Advantages of proposed system:

1. LSTM, GRU, and ARIMA are some of the advanced forecasting models we use in our work. These models are built to work with time series data. This variety lets us look into predicting methods in more depth, which could help us figure out complicated trends in air pollution levels over time.
2. In our work, on the other hand, we use RMSE, MAE, and R-squared as strong measures that are especially designed for evaluating forecasts.
3. We directly deal with the problem of predicting the amount of pollution in the air. This gives us useful information about how much pollution will be in the future, so we can plan ahead for ways to stop it.
4. We clearly take into account changes and trends in the amounts of air pollutants over time. This helps us get a better sense of both short-term and long-term changes, which can make it much easier to guess how much pollution there will be.
5. The use of deep learning methods such as LSTM and GRU in this study makes it possible to find complex time connections in the data. These methods can handle complicated patterns better, which could lead to more accurate predictions.

FUNCTIONAL REQUIREMENTS

1. Data Collection
2. Image processing
3. Data augmentation
4. Training model
5. Final outcome

4.4 NON FUNCTIONAL REQUIREMENTS

To rate the quality of a software system, non-functional requirements (NFR) are used. Some of these needs are speed, ease of use, security, and flexibility. All of these are very important for the system to work well. The speed at which a website loads is an example of a requirement that is not useful. If these conditions aren't met, users might be unhappy. During agile backlogs, non-functional needs could make it hard to build a system. The

website has to load in three seconds if more than 10,000 people are visiting at the same time. Along with functional needs, you should also list non-functional demands.

- Usability requirement
- Serviceability requirement
- Manageability requirement
- Recoverability requirement
- Security requirement
- Data Integrity requirement
- Capacity requirement
- Availability requirement
- Scalability requirement
- Interoperability requirement
- Reliability requirement
- Maintainability requirement
- Regulatory requirement
- Environmental requirement

5. SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE:

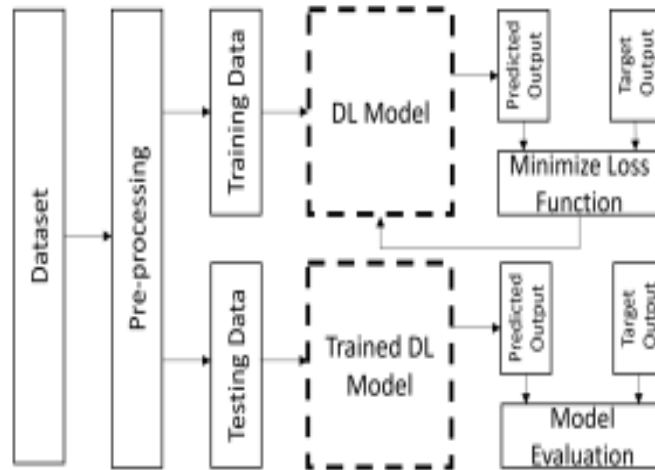


Fig.5.1 System architecture

DATA FLOW DIAGRAM:

1. DFDs are termed bubble charts. Data that enters, is processed, and exits a system may be shown in a simple pictorial model.
2. Data flow diagrams are crucial model tools. We utilize it to model system pieces. These pieces are the system process, its data, an outside body that communicates to it, and information flow.
3. DFD shows how events modify data across the system. It draws visuals of information's input-to-output transformation.
4. Bubble charts are DFD. DFDs may demonstrate system operation at any complexity. DFD has tiers with increasing functionality and information flow

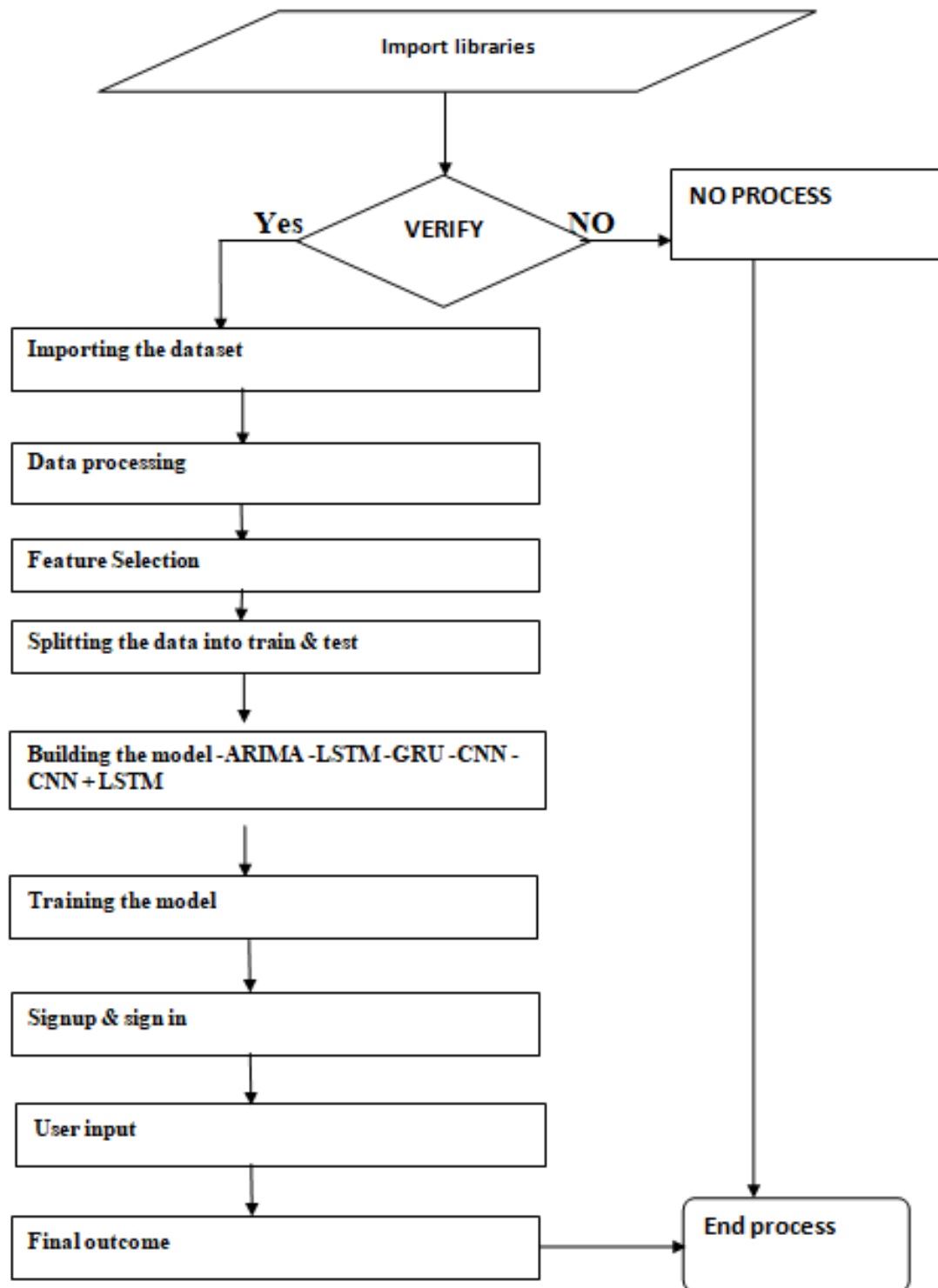


Fig.5.2 Data Flow Diagram

5.2 UML DIAGRAMS

UML is Unified Modeling Language. UML, a standard object-oriented software engineering modeling language, may be used for anything. The Object Management Group produced and manages the standard.

UML aims to become the standard for object-oriented computer program modeling. UML currently has a meta-model and syntax. Future UML additions may include methods or processes. Software systems are defined, visualized, developed, and documented using Unified Modeling Language. Business modeling and non-software systems are possible. The UML and other excellent engineering practices work well for describing large, complex systems.

The UML is essential to object-oriented software development and software production. The UML largely illustrates software project planning using graphics.

GOALS:

The main goals that went into making the UML are listed below:

1. Give people a visual modeling language that is ready to use and creative so that they can make and share important models.
2. Give ways to add to and specialize the core ideas through extendibility and specialization.
3. Not count on a certain computer language or creation process.
4. Give a clear way to understand the language used for models.
5. Help the market for OO tools grow.
6. Back up more advanced ideas in development, like working together, systems, techniques, and components.
7. Use the best methods.

Use case diagram:

UML use case diagrams are behavioural diagrams. A use case analysis defines and creates it. This graphic depicts how a system operates by illustrating its users, their use cases, and how they may interact. Use case map's purpose is to indicate which system elements perform what for each character. Draw scheme participants' roles.

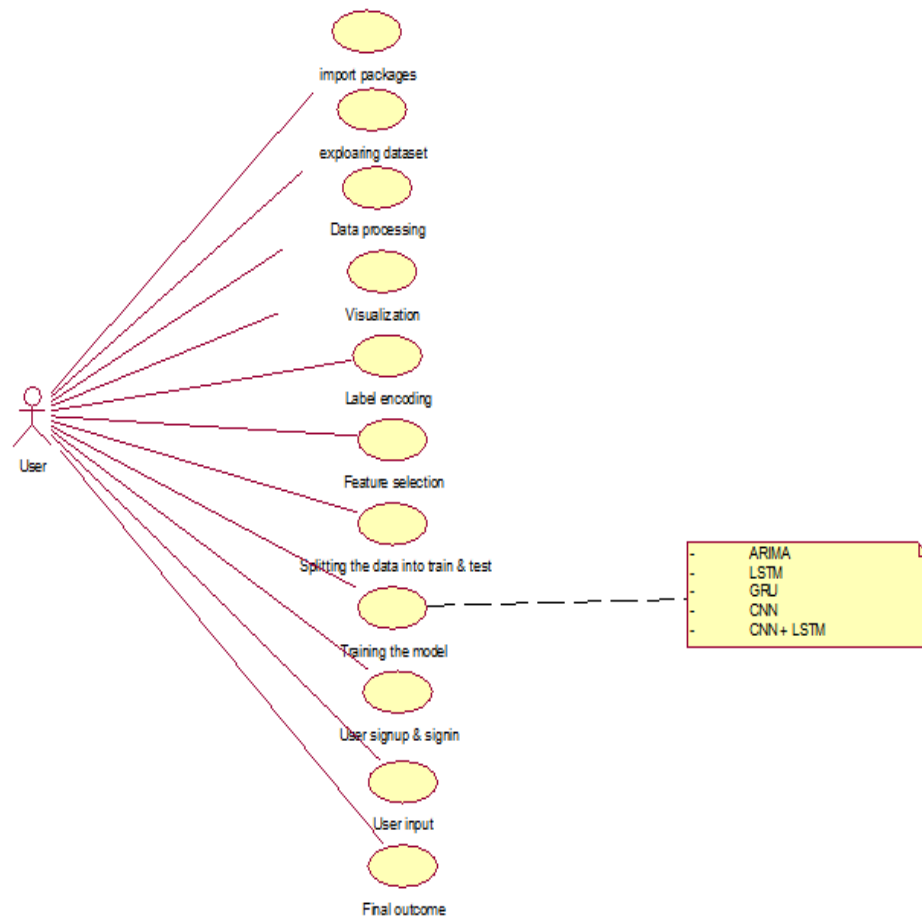


Fig.5.3 Use Case Diagram

Class diagram:

A more accurate use case diagram and detailed system structure planning are achieved using the class diagram. The class diagram groups the use case diagram participants into related classes. All classes may have "is-a" or "has-a" connections. Each class map class may have various abilities. These are class "methods" characteristics. Additionally, each class may have unique "attributes" that distinguish it.

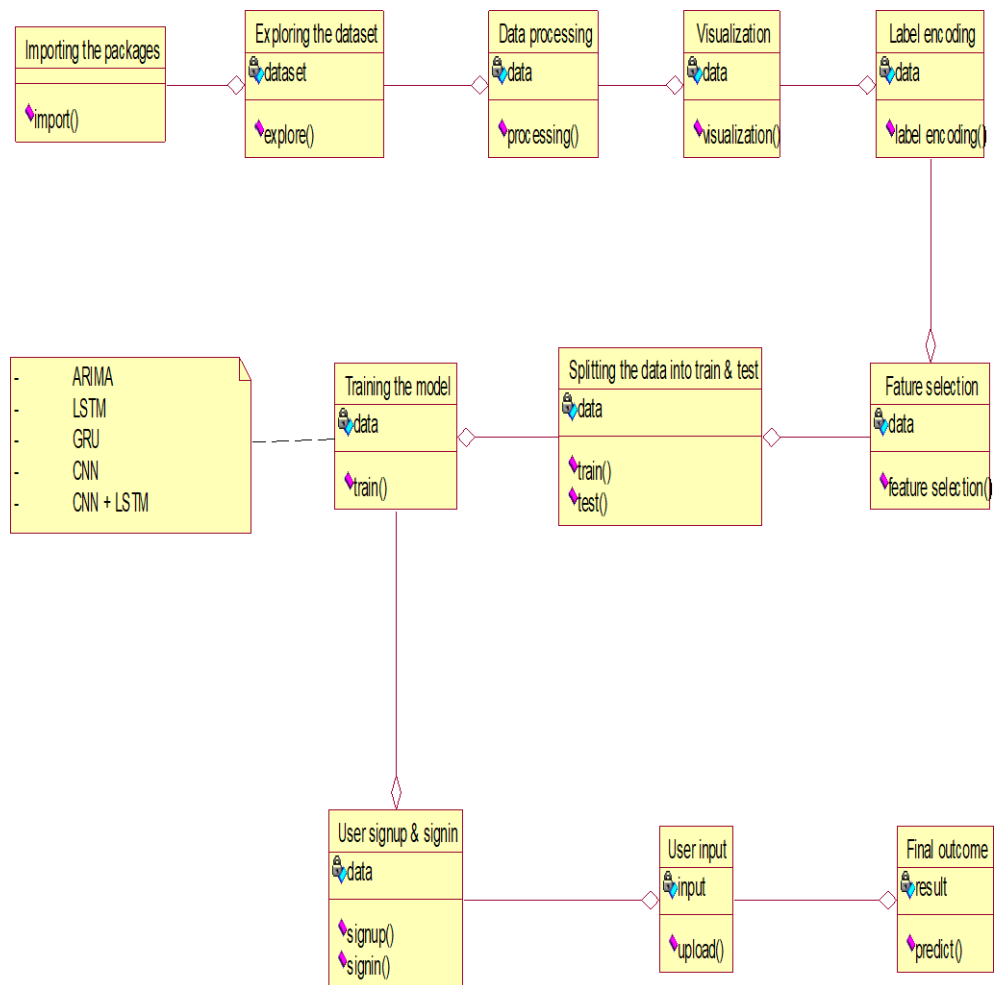


Fig.5.4 Class Diagram

Activity diagram:

A system's process steps are shown in an activity diagram. A state diagram has activities, acts, changes, starting and end states, and guard conditions. An activity diagram has the same things.

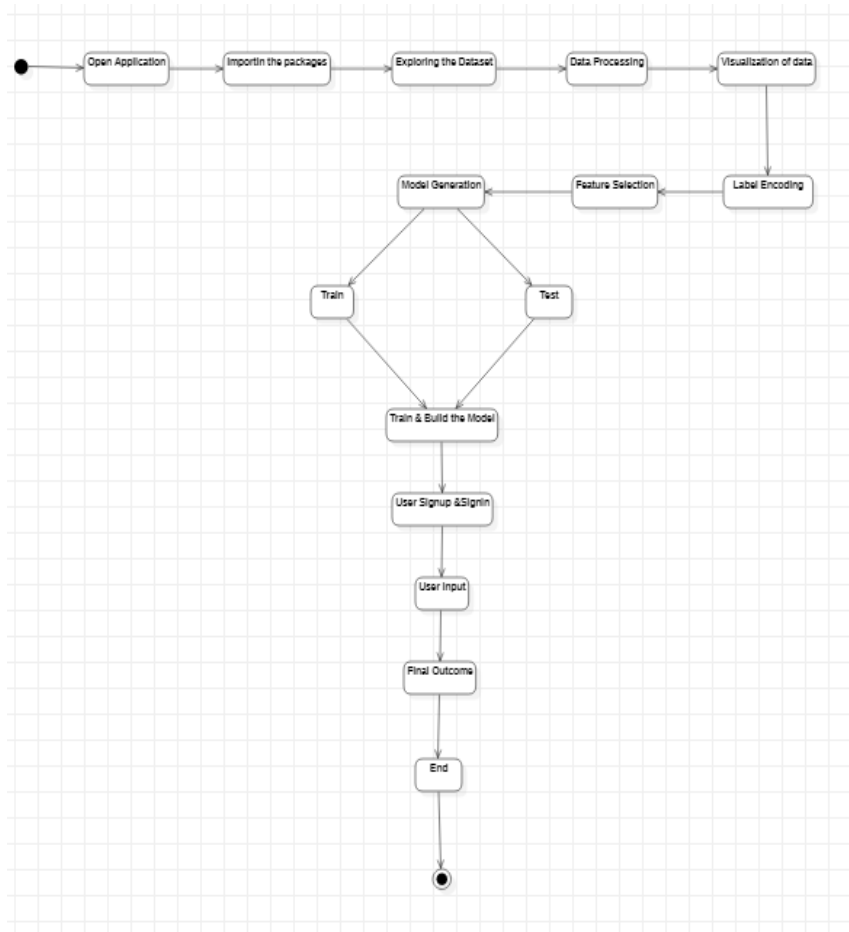


Fig.5.5 Activity Diagram

Sequence diagram:

Sequence diagrams demonstrate system components' interactions. Time order is extremely crucial in series maps. The precise sequencing of item interactions is given step by step. "Messages" are how sequence diagram objects communicate.

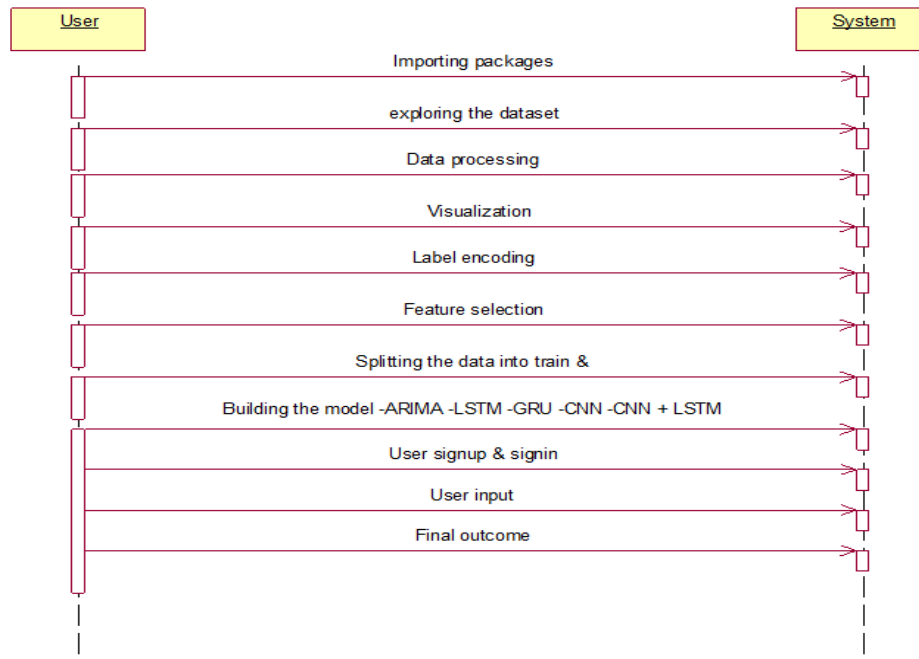


Fig.5.6 Sequence Diagram

Collaboration diagram:

The ways that different things connect with each other are shown in a collaboration image. There is a list of relationships with numbers that make it easier to see how they fit together. It is easier to see all the ways that each object can connect with other objects when you use the cooperation image.

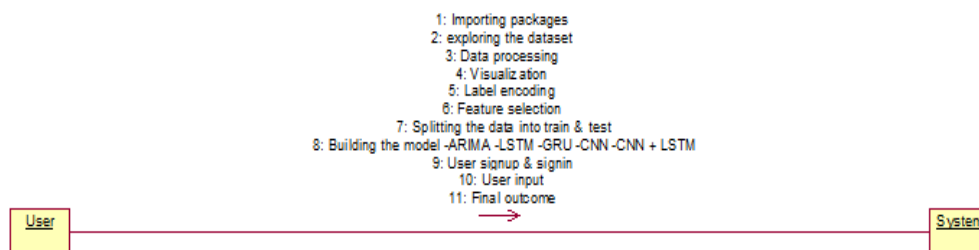


Fig.5.7 Collaboration Diagram

Component diagram:

The component image shows the system's high-level elements. This picture depicts system pieces and their connections at a high level. A component diagram displays system pieces removed after construction or building

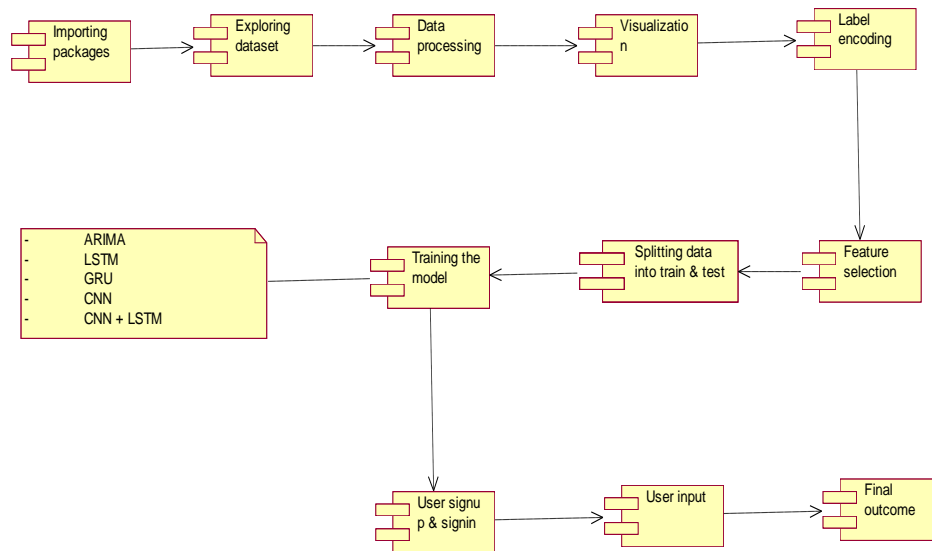


Fig.5.8 Component Diagram

Deployment diagram:

The setup of the application's running parts is shown in the launch picture. If you want to use this model, you should wait until your system is fully built and ready to go.

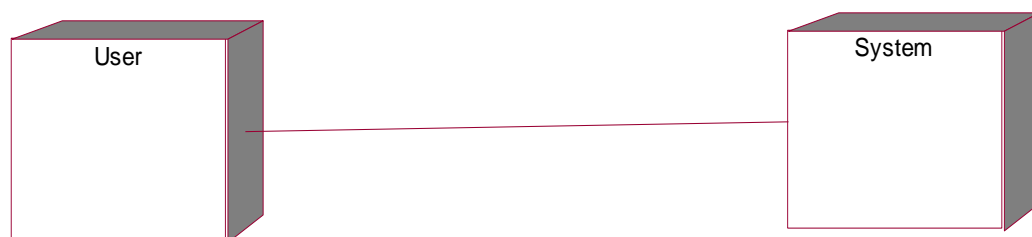


Fig.5.9 Deployment diagram

6. IMPLEMENTATION

MODULES:

- Data exploration: we will use this tool to put data into the system.
- Image processing: We will use the module to turn an image into a digital file and do some actions on it to get useful data.
- Model generation: Building the model -ARIMA -LSTM -GRU -CNN -CNN + LSTM.
- User signup and login: This module will get you registered and logged in.
- User input: This part will be used to give information for forecasts.
- Prediction: the final prediction is shown

Dataset

<https://www.kaggle.com/datasets/cpluzshrijayan/air-quality-prediction-harbor>

Extension

In the base paper, the author suggested using the Air Quality dataset with ARIMA and Deep Learning Models, with DL getting 85% of the r2 score. To make things even better, we used an ensemble method that combined the forecasts of several separate models to make a stronger and more accurate final prediction.

We can improve the results even more, though, by looking into other group methods, like CNN and CNN + LSTM, which got 90% or more of the time.

As an add-on, we can use the Flask framework to build the front end for testing with users and authenticating users.

Algorithms:

ARIMA –ARIMA stands for "autoregressive integrated moving average." It is a type of statistical analysis that uses time series data to help us understand the data set better or guess what trends will happen in the future. Autoregressive statistical models guess what values will be in the future based on values that have already happened. For instance, an ARIMA model might try to guess the future price of a stock by looking at how it has

done in the past or guess how much money a company will make by looking at how much it has made in the past.

LSTM –The LSTM is a special kind of Recurrent Neural Network that can deal with the problem of disappearing gradients that RNNs have. Hochreiter and Schmidhuber came up with LSTM, which fixes the issue that regular RNNs and machine learning algorithms are having. The Keras library can be used to make LSTM work in Python.

GRU –Cho et al. proposed the Gated Recurrent Unit (GRU) in 2014 as an easier option to Long Short-Term Memory (LSTM) networks. The GRU is a type of recurrent neural network (RNN).

CNN – A CNN is a type of network design for deep learning algorithms. It is perfect for jobs that need to recognize images and process pixel data. There are different kinds of neural networks used in deep learning, but CNNs are the best for finding and recognizing things.

CNN + LSTM–Long short-term memory with CNN and LSTM. Recurrent neural networks (RNNs) have been made better by long short-term memory. Instead of regular RNN units, LSTM suggests memory blocks as a way to solve the disappearing and growing gradient problem [51]. The main difference between it and RNNs is that it adds a cell state to store long-term states.

6.2 SAMPLE CODE:

```
In [1]: import warnings
warnings.filterwarnings('ignore')

In [2]: import pandas as pd
import numpy as np
import seaborn as sb

In [3]: from sklearn.model_selection import train_test_split
from statsmodels.tsa.stattools import adfuller
from pandas.plotting import autocorrelation_plot
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import statsmodels.api as sm
#from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.arima_model import ARIMA
from pandas.tseries.offsets import DateOffset

In [4]: data_path = 'Air Quality Prediction Time.csv'
data = pd.read_csv(data_path, engine='c')

In [5]: # convert the 'Date' column to datetime format
data['date'] = pd.to_datetime(data['timestamp'])

# Check the format of 'Date' column
data = data.iloc[:, [1,8]]
data.set_index('date', inplace=True)
data.info()
```

Fig. 6.1

```
In [6]: size=len(data)-int(0.2*len(data))

In [7]: training_data=data.iloc[:size, :]
testing_data = data.iloc[size:, :]
#training_data, testing_data = train_test_split(data, test_size=0.2, random_state=25,shuffle=False)

print(f"No. of training examples: {training_data.shape[0]}")
print(f"No. of testing examples: {testing_data.shape[0]}")

No. of training examples: 14055
No. of testing examples: 3513

In [8]: training_data.plot()
plt.show()
```

```
In [9]: def adfuller_test(timestamp):
result=adfuller(timestamp)
labels = ['ADF Test Statistic', 'p-value', '#Lags Used', 'Number of Observations']

return pd.DataFrame(result[:4], index=labels, columns=['adfuller_test'])

stationary_test=adfuller_test(training_data['particulate_matter'])
stationary_test
```

```
Out[9]:
```

	adfuller_test
ADF Test Statistic	-3.174372
p-value	0.021517
#Lags Used	0.000000
Number of Observations	14054.000000

```
In [10]: autocorrelation_plot(training_data['particulate_matter'].values)
plt.show()
```

Fig. 6.2

```
In [11]: def difference(df,shift):
return data-data.shift(shift)

In [12]: training_data['f_diff'] = difference(training_data['particulate_matter'], 1)

In [13]: adfuller_test_fdiff=adfuller_test(training_data['f_diff'].dropna())

In [14]: adfuller_test_fdiff
```

```
Out[14]:
```

	adfuller_test
ADF Test Statistic	-119.766186
p-value	0.000000
#Lags Used	0.000000
Number of Observations	14053.000000

```
In [15]: def plot_corrs(df):
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df,lags=40,ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df,lags=40,ax=ax2)

In [16]: plot_corrs(training_data['particulate_matter'].values)
```

Fig. 6.3

```

In [17]: history=[h for h in list(training_data['particulate_matter'])]
         future=[f for f in list(testing_data['particulate_matter'])]

In [18]: model1=ARIMA(history, order=(1,0,0))
         model1_fit=model1.fit()
         model1_fit.summary()

In [19]: from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

In [20]: data = pd.read_csv('Air Quality Prediction Cleaned.csv')

In [21]: data.info()

In [24]: data.columns
Out[24]: Index(['s.no', 'City', 'Season', 'Date', 'Ship Entry/Left',
              'Type of ship present', 'PM2.5', 'PM2.5 AQI', 'PM2.5 AQI CAT', 'NO2',
              'NO2 AQI', 'NO2 AQI CAT', 'NH3', 'NH3 AQI', 'NH3 AQI CAT', 'CO',
              'CO AQI', 'CO AQI CAT', 'SO2', 'SO2 AQI', 'SO2 AQI CAT', 'O3', 'O3 AQI',
              'O3 AQI CAT', 'VOC', 'VOC AQI', 'VOC AQI CAT', 'AQI', 'AQI_LVL'],
              dtype='object')

In [25]: data = data[['Season', 'PM2.5', 'PM2.5 AQI', 'NO2',
                    'NO2 AQI', 'NH3', 'NH3 AQI', 'CO', 'CO AQI', 'SO2', 'SO2 AQI',
                    'O3', 'O3 AQI', 'VOC', 'VOC AQI', 'AQI', 'AQI_LVL']]

In [26]: data.head()
Out[26]:
   Season  PM2.5  PM2.5 AQI  NO2  NO2 AQI  NH3  NH3 AQI  CO  CO AQI  SO2  SO2 AQI  O3  O3 AQI  VOC  VOC AQI  AQI  AQI_LVL
0  Spring    6.55        27  0.90    1  0.02    1  0.11    1  1.31    1  1.88    1  0.05    1  4.714286  GOOD
1  Spring   12.58        52  1.25    1  0.09    1  0.13    1  1.55    1  2.83    2  0.06    1  8.428571  GOOD
2  Spring   25.98        80  1.60    1  0.21    3  0.55    6  1.89    1  3.22    3  0.09    1  13.571429  GOOD
3  Spring   29.88        88  1.56    1  0.26    3  0.75    8  2.66    3  3.95    3  0.07    1  15.285714  GOOD
4  Spring   35.93       102  2.65    2  0.34    4  0.84    9  3.66    4  4.58    4  0.05    1  18.000000  GOOD

In [27]: data.info()

```

Fig. 6.4

```

In [30]: # Import Label encoder
         from sklearn import preprocessing

         # Label_encoder object knows
         # how to understand word Labels.
         label_encoder = preprocessing.LabelEncoder()

         # Encode Labels in column 'species'.
         data['Season'] = label_encoder.fit_transform(data['Season'])
         data['AQI_LVL'] = label_encoder.fit_transform(data['AQI_LVL'])

         data['Season'].unique()

Out[30]: array([0, 1], dtype=int64)

In [31]: X = data.drop(['AQI_LVL'], axis=1)
         y = data['AQI_LVL']

In [32]: # splitting the dataset 80% for training and 20% testing
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

```

Fig. 6.5

```

In [33]: from tensorflow.keras.callbacks import EarlyStopping
        from tensorflow.keras.callbacks import ModelCheckpoint
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import classification_report, confusion_matrix
        from tensorflow.keras.models import Sequential
        from keras.utils import np_utils
        from tensorflow.keras.layers import LSTM, GRU, Dense, Embedding, Dropout, Input, Attention, Layer, Concatenate, Permute, Dot, Multiply, Flatten
        from tensorflow.keras.layers import RepeatVector, Dense, Activation, Lambda
        from tensorflow.keras.models import Sequential
        from tensorflow.keras import backend as K, regularizers, Model, metrics
        from tensorflow.keras.backend import cast

In [34]: X_train=X_train.values
        X_test=X_test.values

In [35]: # design network
        np.random.seed(7)

        # X_train et X_val sont des dataframe qui contient Les features
        train_X=X_train
        val_X=X_test

In [36]: ## Reshape input to be 3D [samples, timesteps, features] (format requis par LSTM)
        train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
        val_X = val_X.reshape((val_X.shape[0], 1, val_X.shape[1]))

In [37]: ## Recuperation des Labels
        train_y=y_train
        val_y=y_test

```

Fig. 6.6

```

In [38]: class attention(Layer):
        def __init__(self, **kwargs):
            super(attention, self).__init__(**kwargs)

        def build(self, input_shape):
            self.W=self.add_weight(name="att_weight",shape=(input_shape[-1],1),initializer="normal")
            self.b=self.add_weight(name="att_bias",shape=(input_shape[1],1),initializer="zeros")
            super(attention, self).build(input_shape)

        def call(self,x):
            et=K.squeeze(K.tanh(K.dot(x,self.W)+self.b),axis=-1)
            at=K.softmax(et)
            at=K.expand_dims(at,axis=-1)
            output=x*at
            return K.sum(output,axis=1)

        def compute_output_shape(self,input_shape):
            return (input_shape[0],input_shape[-1])

        def get_config(self):
            return super(attention,self).get_config()

In [39]: inputs1=Input((1,16))
        att_in=LSTM(50,return_sequences=True,dropout=0.3,recurrent_dropout=0.2)(inputs1)
        att_in_1=LSTM(50,return_sequences=True,dropout=0.3,recurrent_dropout=0.2)(att_in)
        att_out=attention()(att_in_1)
        outputs1=Dense(1,activation='sigmoid',trainable=True)(att_out)
        model1=Model(inputs1,outputs1)

In [40]: model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

In [41]: history=model1.fit(train_X, train_y,epochs=10,batch_size=2)

```

Fig. 6.7

```

In [42]: y_pred = model1.predict(val_X, verbose=1)
         y_pred = np.argmax(y_pred,axis=1)

         1/1 [=====] - 1s 582ms/step

In [43]: lstm_acc = accuracy_score(y_pred, val_y)
         lstm_prec = precision_score(y_pred, val_y,average='weighted')
         lstm_rec = recall_score(y_pred, val_y,average='weighted')
         lstm_f1 = f1_score(y_pred, val_y,average='weighted')

In [71]: storeResults('LSTM',lstm_acc,lstm_prec,lstm_rec,lstm_f1)

In [44]: from matplotlib import pyplot

In [46]: # Plot of accuracy vs epoch for train and test dataset
         plt.plot(history.history['loss'])
         plt.plot(history.history['accuracy'])
         plt.title("Plot of accuracy vs epoch for train and test dataset")
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.show()

In [47]: inputs1=Input((1,16))
         att_in=GRU(50,return_sequences=True,dropout=0.3,recurrent_dropout=0.2)(inputs1)
         att_in_1=GRU(50,return_sequences=True,dropout=0.3,recurrent_dropout=0.2)(att_in)
         att_out=attention()(att_in_1)
         outputs1=Dense(1,activation='sigmoid',trainable=True)(att_out)
         model1=Model(inputs1,outputs1)

In [48]: model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])

In [49]: history=model1.fit(train_X, train_y,epochs=10,batch_size=2)

```

Fig. 6.8

```

In [50]: y_pred = model1.predict(val_X, verbose=1)
         y_pred = np.argmax(y_pred,axis=1)

         1/1 [=====] - 0s 339ms/step

In [51]: gru_acc = accuracy_score(y_pred, val_y)
         gru_prec = precision_score(y_pred, val_y,average='weighted')
         gru_rec = recall_score(y_pred, val_y,average='weighted')
         gru_f1 = f1_score(y_pred, val_y,average='weighted')

In [72]: storeResults('GRU',gru_acc,gru_prec,gru_rec,gru_f1)

In [52]: # plot loss during training
         pyplot.subplot(211)
         pyplot.title('Loss')
         pyplot.plot(history.history['loss'], label='train')
         pyplot.plot(history.history['acc'], label='train')
         pyplot.legend()
         # plot accuracy during training
         pyplot.show()

```

Fig. 6.9


```

In [53]: from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense
        from tensorflow.keras.models import Model, load_model
        from tensorflow.keras.utils import to_categorical
        from tensorflow.keras.layers import Dropout
        from tensorflow.keras.layers import Flatten
        from tensorflow.keras.layers import Conv1D
        from tensorflow.keras.layers import MaxPooling1D

        verbose, epoch, batch_size = 1, 10, 2
        activationFunction='relu'

In [54]: X_train = X_train.reshape(-1, X_train.shape[1],1)
        X_test = X_test.reshape(-1, X_test.shape[1],1)

        Y_train=to_categorical(y_train)
        Y_test=to_categorical(y_test)

In [55]: def CNN():

        cnnmodel = Sequential()
        cnnmodel.add(Conv1D(filters=128, kernel_size=2, activation='relu',input_shape=(X_train.shape[1],X_train.shape[2])))
        cnnmodel.add(MaxPooling1D(pool_size=2))
        cnnmodel.add(Dropout(rate=0.2))
        cnnmodel.add(Flatten())
        cnnmodel.add(Dense(3, activation='softmax'))
        cnnmodel.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
        cnnmodel.summary()
        return cnnmodel

cnnmodel = CNN()

```

Fig. 6.10

```

In [56]: modelhistory = cnnmodel.fit(X_train, Y_train, epochs=10, verbose=verbose, validation_split=0.2, batch_size = batch_size)

Epoch 1/10
38/38 [=====] - 0s 4ms/step - loss: 4.3388 - accuracy: 0.5658 - val_loss: 0.1532 - val_accuracy: 0.9000
Epoch 2/10
38/38 [=====] - 0s 1ms/step - loss: 1.9881 - accuracy: 0.6711 - val_loss: 0.2500 - val_accuracy: 0.9000
Epoch 3/10
38/38 [=====] - 0s 1ms/step - loss: 2.5463 - accuracy: 0.6579 - val_loss: 1.6805 - val_accuracy: 0.8000
Epoch 4/10
38/38 [=====] - 0s 1ms/step - loss: 2.1278 - accuracy: 0.7368 - val_loss: 0.2534 - val_accuracy: 0.9500
Epoch 5/10
38/38 [=====] - 0s 1ms/step - loss: 1.7949 - accuracy: 0.8026 - val_loss: 0.4499 - val_accuracy: 0.9500
Epoch 6/10
38/38 [=====] - 0s 1ms/step - loss: 0.8767 - accuracy: 0.8289 - val_loss: 0.8243 - val_accuracy: 0.8500
Epoch 7/10
38/38 [=====] - 0s 1ms/step - loss: 0.9423 - accuracy: 0.8158 - val_loss: 0.5044 - val_accuracy: 0.8500
Epoch 8/10
38/38 [=====] - 0s 1ms/step - loss: 1.1635 - accuracy: 0.7895 - val_loss: 0.3816 - val_accuracy: 0.9500
Epoch 9/10
38/38 [=====] - 0s 1ms/step - loss: 0.1620 - accuracy: 0.9474 - val_loss: 0.7572 - val_accuracy: 0.9500
Epoch 10/10
38/38 [=====] - 0s 1ms/step - loss: 1.0619 - accuracy: 0.8553 - val_loss: 0.2949 - val_accuracy: 0.9000

```

Fig. 6.11

```
In [57]: # Plot of accuracy vs epoch for train and test dataset
plt.plot(modelhistory.history['accuracy'])
plt.plot(modelhistory.history['val_accuracy'])
plt.title("Plot of accuracy vs epoch for train and test dataset")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.show()

# Plot of Loss vs epoch for train and test dataset
plt.plot(modelhistory.history['loss'])
plt.plot(modelhistory.history['val_loss'])
plt.title("Plot of loss vs epoch for train and test dataset")
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()

cnnpredictions = cnnmodel.predict(X_test, verbose=1)
cnn_predict=np.argmax(cnnpredictions,axis=1)

y_pred = cnnmodel.predict(X_test, verbose=1)
y_pred = np.argmax(y_pred,axis=1)
```

Fig.6.12

```
In [60]: import tensorflow as tf
tf.keras.backend.clear_session()

model_en = tf.keras.models.Sequential([tf.keras.layers.Conv1D(filters=64, kernel_size=5, strides=1, padding="causal", activation="relu", input_shape=(X_train.shape[1], X_train.shape[2])),
tf.keras.layers.MaxPooling1D(pool_size=2, strides=1, padding="valid"),
tf.keras.layers.Conv1D(filters=32, kernel_size=3, strides=1, padding="causal", activation="relu"),
tf.keras.layers.MaxPooling1D(pool_size=2, strides=1, padding="valid"),
tf.keras.layers.LSTM(128, return_sequences=True),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(128, activation="relu"),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(32, activation="relu"),
tf.keras.layers.Dropout(0.1),
tf.keras.layers.Dense(3)
])

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(5e-4,
                                                              decay_steps=1000000,
                                                              decay_rate=0.98,
                                                              staircase=False)

model_en.compile(loss=tf.keras.losses.MeanSquaredError(),
                 optimizer=tf.keras.optimizers.SGD(learning_rate=lr_schedule, momentum=0.8),
                 metrics=['accuracy'])
model_en.summary()
```

Fig.6.13

```
In [61]: modelhistory = model_en.fit(X_train, Y_train, epochs=10, verbose=verbose, validation_split=0.2, batch_size = batch_size)

Epoch 1/10
38/38 [=====] - 2s 13ms/step - loss: 0.3069 - accuracy: 0.6053 - val_loss: 0.1493 - val_accuracy: 0.6000
Epoch 2/10
38/38 [=====] - 0s 5ms/step - loss: 0.1591 - accuracy: 0.7237 - val_loss: 0.1029 - val_accuracy: 0.9500
Epoch 3/10
38/38 [=====] - 0s 5ms/step - loss: 0.1350 - accuracy: 0.8026 - val_loss: 0.0888 - val_accuracy: 0.8500
Epoch 4/10
38/38 [=====] - 0s 5ms/step - loss: 0.1145 - accuracy: 0.8289 - val_loss: 0.0918 - val_accuracy: 0.9500
Epoch 5/10
38/38 [=====] - 0s 5ms/step - loss: 0.0991 - accuracy: 0.8684 - val_loss: 0.0805 - val_accuracy: 0.9500
Epoch 6/10
38/38 [=====] - 0s 5ms/step - loss: 0.0833 - accuracy: 0.9211 - val_loss: 0.0644 - val_accuracy: 0.9000
Epoch 7/10
38/38 [=====] - 0s 5ms/step - loss: 0.0887 - accuracy: 0.8816 - val_loss: 0.0555 - val_accuracy: 0.9000
Epoch 8/10
38/38 [=====] - 0s 5ms/step - loss: 0.0779 - accuracy: 0.9211 - val_loss: 0.0530 - val_accuracy: 0.9500
Epoch 9/10
38/38 [=====] - 0s 5ms/step - loss: 0.0745 - accuracy: 0.9474 - val_loss: 0.0578 - val_accuracy: 0.9500
Epoch 10/10
38/38 [=====] - 0s 6ms/step - loss: 0.0638 - accuracy: 0.9737 - val_loss: 0.0539 - val_accuracy: 0.9000
```

Fig.6.14

```

In [62]: # Plot of accuracy vs epoch for train and test dataset
plt.plot(modelhistory.history['accuracy'])
plt.plot(modelhistory.history['val_accuracy'])
plt.title("Plot of accuracy vs epoch for train and test dataset")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.show()

# Plot of Loss vs epoch for train and test dataset
plt.plot(modelhistory.history['loss'])
plt.plot(modelhistory.history['val_loss'])
plt.title("Plot of loss vs epoch for train and test dataset")
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()

```

Fig.6.15

```

In [82]: #creating dataframe
result = pd.DataFrame({ 'ML Model' : ML_Model,
                        'Accuracy' : accuracy,
                        'Precision': precision,
                        'Recall'   : recall,
                        'F1-Score': f1score,

                        })

```

In [83]: result

Out[83]:

	ML Model	Accuracy	Precision	Recall	F1-Score
0	LSTM	0.120	1.000	0.120	0.214
1	GRU	0.120	1.000	0.120	0.214
2	CNN	0.960	0.964	0.960	0.960
3	CNN+LSTM	0.979	0.979	0.979	0.979

Fig.6.16

7. OVERVIEW OF TECHNOLOGIES

DEEP LEARNING:

What is deep learning?

A three-level neural network is deep learning in machine learning. Though far from perfect, these neural networks strive to replicate the brain. This enables them "learn" from plenty of data. A neural network with one layer may generate imprecise estimates, but additional hidden layers can enhance accuracy.

Many AI applications and services employ deep learning to improve automation by executing physical and mental tasks without human assistance. Digital assistants, voice-enabled TV remotes, and credit card fraud detection employ deep learning. Self-driving vehicles employ deep learning.

Deep learning vs. machine learning

How are deep learning and machine learning different? Deep learning utilizes different data and learns differently from typical machine learning.

Structured, labelled data helps machine learning algorithms predict. This involves describing and tabulating model features from raw data. Unstructured data is frequently pre-processed to arrange it. It utilizes unstructured data sometimes.

Deep learning does not need all the data pre-processing of machine learning. These algorithms handle unstructured text and images. Experts are required less since they can automate feature extraction. Suppose we wanted to arrange pet photos into categories like "cat," "dog," "hamster," etc. Deep learning algorithms can determine which animal features, such ears, are most distinguishing. Experts manually rank this list of qualities in machine learning.

The deep learning algorithm then uses gradient descent and backpropagation to enhance accuracy. This helps it predict a new animal photo better.

Machine learning and deep learning models learn differently. Guided, uncontrolled, and reinforcement learning. Labelled datasets for supervised learning to categorize or predict need human involvement to name the incoming data. Unsupervised learning doesn't need named datasets. Instead, it finds data patterns and organizes them by distinct qualities. Reinforcement learning improves a model's performance in a particular situation by providing feedback. Win the largest reward.

"AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?" discusses the small variations between the technologies.

"Supervised vs. Unsupervised Learning: What's the Difference?" explains the distinctions.

How deep learning works

Deep learning neural networks strive to mimic the human brain using data sources, weights, and bias. These sections collaborate to discover, organize, and explain data.

Many layers of connected nodes make up deep neural networks. Layers enhance prediction and categorization by building on each other. Results spread over a network via forward propagation. Deep neural networks have input and output layers. Data is entered and processed at the output layer by the deep learning model. The final prediction or categorization is produced in the output layer.

Backpropagation trains a model by moving backwards through the levels and modifying function weights and biases depending on prior estimate errors. Gradient descent is used for this. Forward propagation and backpropagation assist neural networks predict and correct errors. The software improves with time.

The simplest deep neural network requires no more explanation. Deep learning techniques are sophisticated, and different neural networks are utilized for different tasks or data sets. For instance,

- CNNs, often used in computer vision and image classification, can identify characteristics and patterns in images, enabling object recognition and identification. CNN defeated a human in object recognition for the first time in 2015.

- Recurrent neural networks (RNNs) are often utilized in natural language and voice recognition applications due to their ability to incorporate linear or time series data.

Deep learning Applications examples

We use deep learning every day, but most of the time, users don't even notice that complicated data processing is going on in the background because the apps are so well built into goods and services. The following are some examples of these:

Law enforcement

Deep learning systems can look at financial data and learn from it to find risky trends that could mean fraud or other illegal behavior. By finding patterns and evidence in audio and video recordings, pictures, and documents, speech recognition, computer vision, and other deep learning applications can make investigative work faster and better. This is because they can handle large amounts of data more accurately and quickly.

Financial services

Predictive analytics is often used by financial companies to help clients handle their credit and investment accounts, trade stocks automatically, and decide whether to lend money to a business.

Customer service

An increasing number of businesses use deep learning technology to help with customer service. A simple example of AI is the chatbot, which is used in many apps, services, and customer service sites. Traditional chatbots use natural language and even image recognition, which you can see in choices that look like call centers. Chatbots that are smarter, on the other hand, try to learn whether there is more than one answer to an unclear question. The robot then tries to answer these questions directly or send the conversation to a real person based on the answers it gets.

The idea of a robot is expanded by virtual helpers like Apple's Siri, Amazon's Alexa, or Google Assistant, which can recognize speech. This gives you a new way to interact with people in a personalized way.

Healthcare

Deep learning has been very helpful for the healthcare business ever since medical data and pictures were digitized. Medical imaging experts and doctors can use image recognition software to help them look at and judge more pictures in less time.

Deep learning hardware requirements

A huge amount of computer power is needed for deep learning. High-performance graphical processing units (GPUs) are best because they have a lot of memory and can handle a lot of operations on multiple parts. Managing multiple GPUs on-premises, on the other hand, can put a lot of stress on internal resources and be very expensive to grow.

PYTHON LANGUAGE:

Python is a dynamic, object-oriented language that is ideal for Rapid Application Development due to its high-level data structures, dynamic types, and dynamic binding capabilities. It may be used to program or connect items. Simple Python syntax is easy to learn. It focuses on readability, which lowers the cost of maintaining programs. Python lets you use modules and packages, which makes it easier to break up programs into smaller pieces and reuse code. For all major systems, you can get the Python engine and the large standard library for free in code or binary form, and you can share them with anyone else. A lot of the time, coders fall in love with Python because it helps them get more done. The loop of change, test, and fix is very fast because there is no assembly step. It's easy to find bugs in Python programs because a segmentation fault can't be caused by a bug or bad data. Instead, an exception is raised by the translator when it finds a mistake. A stack trace is shown by the processor when the computer doesn't catch the bad code. A source level debugger in Python allows you to examine variables, execute code, set breakpoints, step through code line by line, and perform other tasks.

Writing code for the debugger in Python is an example of self-analysis. Adding print lines to source code is often the easiest way to make a program better because it makes changing, testing, and fixing go faster. Python is a dynamic, high-level, free, open-source computer language that lets you write both object-oriented and procedural code. Since it doesn't need variable types, x could be either a string or a number.

Features in Python:

Python is a programming language with many functions. Here are some of them:

1. Open Source and Free

Python is available for free on its website. Click the get Python phrase below and the link to get it. [Get Python here](#). Open-source implies anybody may view the source code. Thus, you may get, utilize, and share it.

2. Easy to code

Python is a high-level language. C, C#, JavaScript, Java, etc. are tougher to learn than Python. Anyone can learn Python fundamentals in hours or days. The language is easy. The language is very simple for developers.

3. Easy to Read

You will see that it's easy to learn Python. As was already said, Python's grammar is very easy to understand. Instead of semicolons or braces, the indentations show what the code block is.

4. Object-Oriented Language

Object-oriented computing is one of the most important parts of Python. Python works with object-oriented languages and the ideas of classes, isolation, and so on.

5. GUI Programming Support

Graphical In Python, you can use a package like PyQt5, PyQt4, wxPython, or Tk to make a user interface. The most popular way to use Python to make graphics apps is with PyQt5.

6.High-Level Language

Python is advanced. Python simplifies program writing without requiring system or memory knowledge.

7. Extensible feature

Python is extensible. Python code may be written in C or C++ and built in those languages.

8. Easy to Debug

Great expertise for detecting errors. After reading Python's error traces, you can easily detect and repair most software issues. The code's purpose is obvious at a glance.

9. Python is a Portable language

Python may be used anywhere. Python code written for Windows will function on Linux, Unix, and Mac without modification.

10. Python is an integrated language

Python is also a combined language, which means it's easy to use with other languages like C, C++, and so on.

11. Interpreted Language:

Python executes code line by line, making it interpreted. Unlike C, C++, Java, and other languages, Python code doesn't require compilation. This simplifies code bug detection. Python source code is converted to fast byte code.

12. Large Standard Library

Python has a large standard library of modules and functions, so you don't have to create everything. Python includes packages for regular expressions, unit tests, web tools, and more.

13. Dynamically Typed Language

Python types vary dynamically. Variable types (int, double, long, etc.) are selected during runtime, not beforehand. This means we don't need to explain the variable's type.

14. Frontend and backend development

Use basic tags like `<py-script>` and `<py-env>` to execute and write Python programs in HTML. Your new project is py script. This lets you develop JavaScript-like webpages using Python. Python excels at back-end tasks, and Django and Flask make it easier.

15. Allocating Memory Dynamically

Python doesn't need data type. A variable gets memory instantly when you give it a number at runtime. If y equals 15, developers don't need to put `int y = 18`. Just write 18 for y.

LIBRARIES/PACKAGES:

Tensor flow

This dataflow and differentiable computing tool is free and open source, and it can be used for many things. The tool is used for math with signs. For neural networks and other jobs that use machine learning. Google does work and school on it.

The Google Brain team made TensorFlow just for people who work at Google. It came out on November 9, 2015, under Apache 2.0.

Numpy

You can do a lot of different things with the array package Numpy. This class gives you both a fast multidimensional array object and tools for working with them. You need this package to use Python for science computing. There are many parts to it, but these are the most important ones:

The object is a complex N-dimensional collection with methods for combining C/C++ and Fortran code. It has features that are useful for linear algebra, Fourier transform, and random numbers. Numpy has many science uses and can store general data in a number of different ways. Because it can explain any data type, Numpy can connect to a lot of applications right away.

Pandas

Pandas is an open-source Python tool that has strong data structures that make it easy to work with and examine big amounts of data. Python was used most of the time to get data ready and load it. It didn't really help with the study of the facts. Pants were able to solve this issue. Without a doubt, Pandas lets us handle and study data from any source by letting us load, prepare, edit, model, and examine it. Some of the business and scholarly fields that use Python and Pandas are finance, economics, statistics, analytics, and more.

Matplotlib

Matplotlib is a Python tool for making 2D charts. It can make graphs that can be printed in a variety of forms and used interactively on multiple computers. You can use Python scripts, Python and IPython tools, Jupyter Notebook, four GUI toolkits, and web application servers with this package. To make tough tasks possible and easy tasks simple. You can make bar charts, error charts, scatter plots, histograms, and power spectra with just a few lines of code. You can get ideas from the sample plots and pictures.

The pyplot package looks like MATLAB and lets you make simple plots when you use it with IPython. Power users in MATLAB know how to use a set of tools that let them change things like line styles, font properties, axes properties, and more.

Scikit – learn

The Scikit-learn library gives you a consistent way to use a variety of supervised and unsupervised learning algorithms in Python. It comes with a simple, permissive BSD license and is included in many Linux distributions, which makes it easy for businesses and schools to use.

8. SYSTEM TESTING

Quality assurance (QA) is the name of the group that checks how well an app's different parts work together. We test the system when we do this. There are other names for this type of testing, such as system-level testing and system functionality testing. Sure, a program is checked to make sure it does its job. The main goal of this "black box" test is to find out how the app works. For example, system testing could check that all input from users makes the whole program do what it's meant to do.

Phases of system testing

For this part of the test, a movie guide. Tests are done on each part of a program to make sure the whole thing works well. Most of the time, a quality assurance team tests the system after functional or user-story testing is done on each section and then integration testing is done on each part.

It is put through acceptance testing to make sure it meets all the requirements set by system testing. After that, it is put into production, which is where people use it. The people who are making apps keep track of all the mistakes and decide how many and what kind are okay.

8.1 Software Testing Strategies:

Improvements to the way tests are given are the best way to get the most out of them. A software testing plan tells you what needs to be done, when it needs to be done, and how it should be done. The following software testing methods, as well as combinations of them, are often used to reach this main goal:

Static Testing:

It is called static testing when you look at something without using it. This is done early on in the development process. Desk-checking is mostly done to find mistakes and problems in the code itself. People should do this kind of testing before putting the software into use. It helps keep issues from happening that could be caused by mistakes in the code or issues with how the software was made.

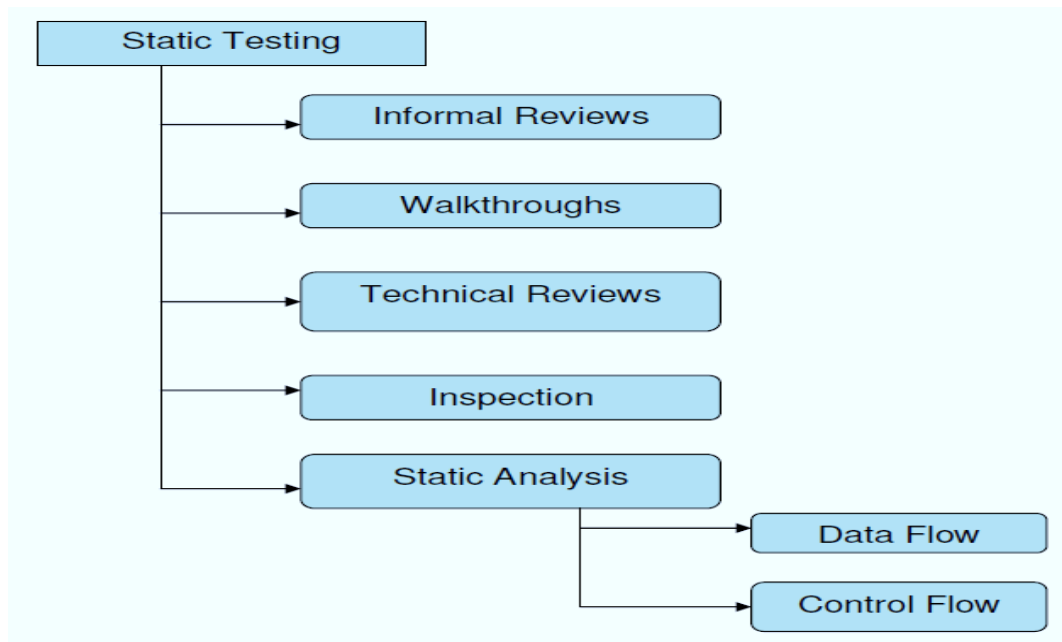


Fig.8.1 Static Testing

Structural Testing:

Test software that hasn't been run yet. Bugs and mistakes must be found and fixed while the program is being developed before it goes into production. So this is structure testing, also known as "white-box testing." Depending on how the software is structured, regression testing is used for unit testing. Often, it's done automatically in the test automation system to speed up development. Because they can see the whole software and how data moves, development and quality assurance engineers can keep an eye on changes to the system (mutation testing) by comparing the most recent test results to those from earlier versions (control flow testing).

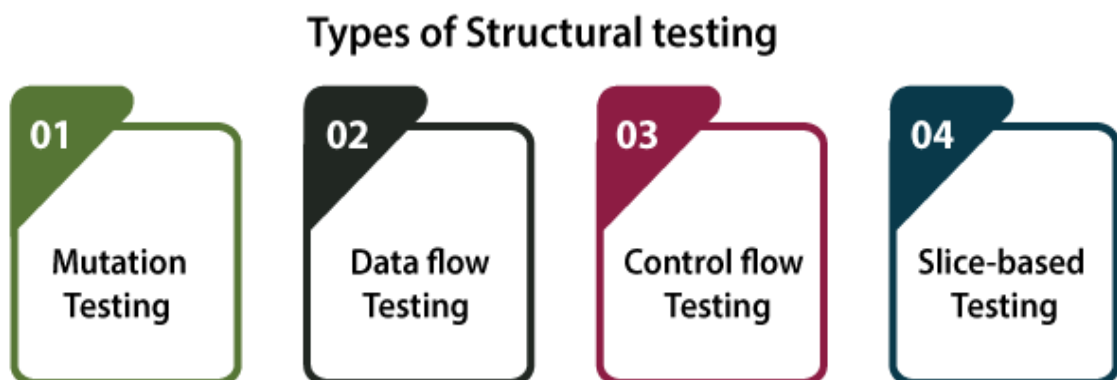


Fig.8.2 Structural Testing

Behavioural Testing:

In the last part of testing, the focus is on how the software acts on jobs instead of how it handles them. So, behavioural testing, which is also sometimes called "black-box testing," includes a number of human tests that look at how the product works from the user's point of view. If quality assurance engineers want to figure out how useful a product is and fix bugs like regular users, they often need to know a lot about the business or other uses of "the black box." Regression tests may be used in behavioral testing to get rid of mistakes people make when doing the same thing over and over. It's possible that you'll have to fill out 100 online forms just to test the goods.

Black Box Testing

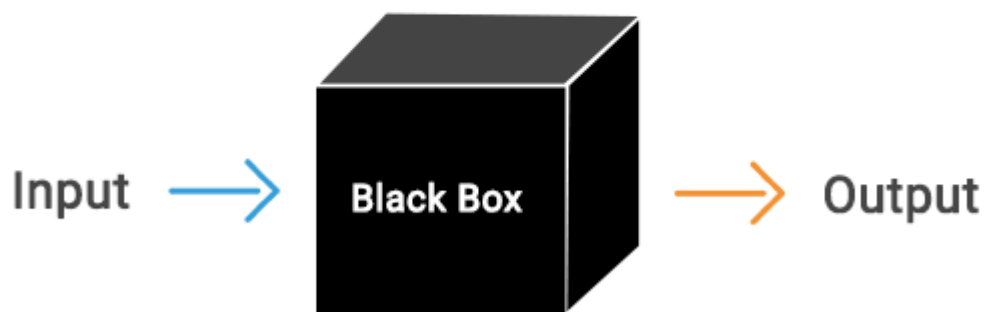


Fig.8.3 Behavioural Testing

8.2 TEST CASES:

S.NO	INPUT	If available	If not available
1	User signup	User get registered into the application	There is no process
2	User sign in	User get login into the application	There is no process
3	Enter input for prediction	Prediction result displayed	There is no process

9. FRONTEND SCREENS

SCREENS:

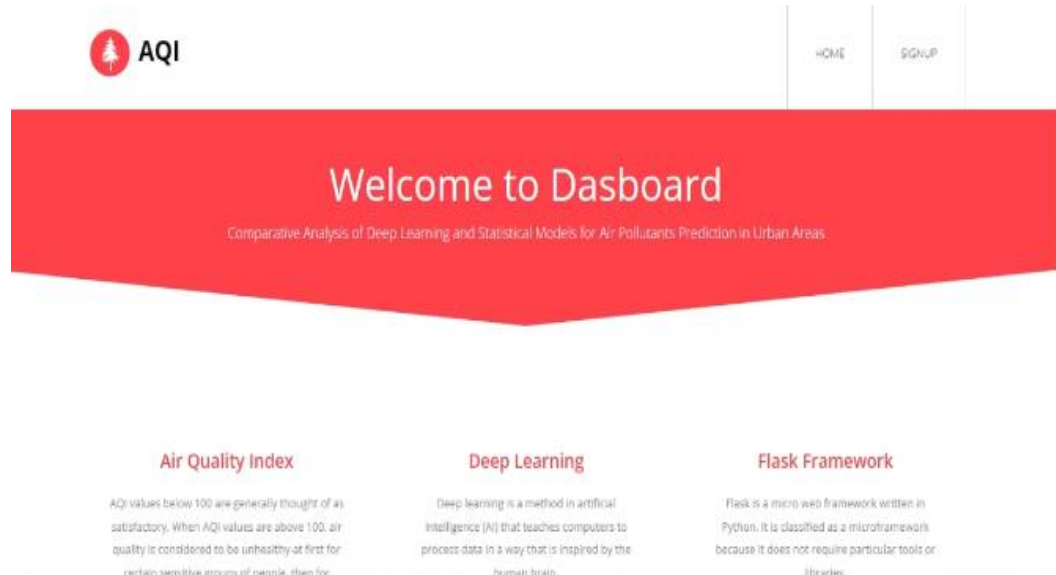


Fig.9.1 Dashboard

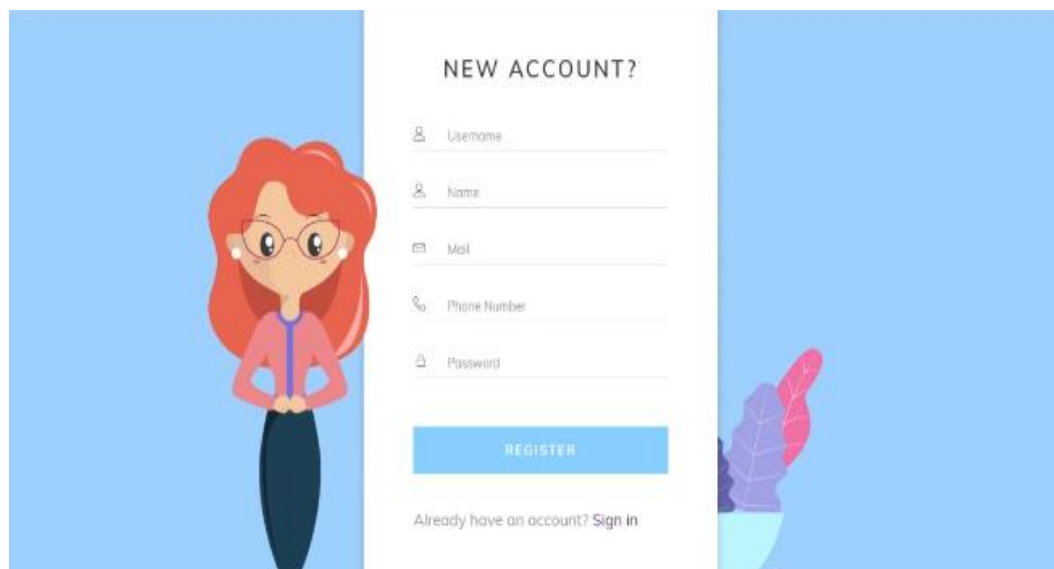


Fig.9.2 Log In Page

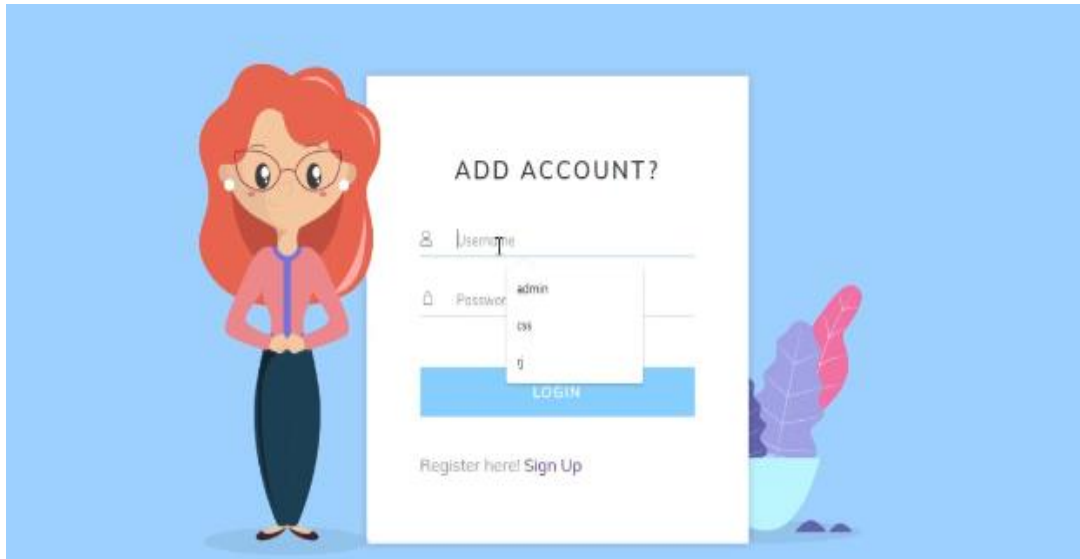


Fig.9.3 Log In Page

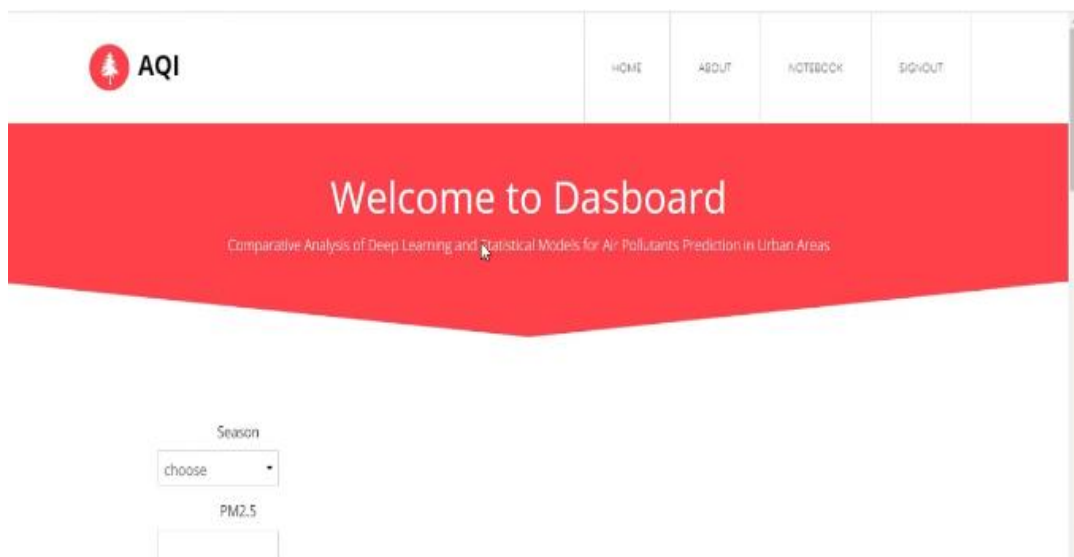


Fig.9.4 User Input Page

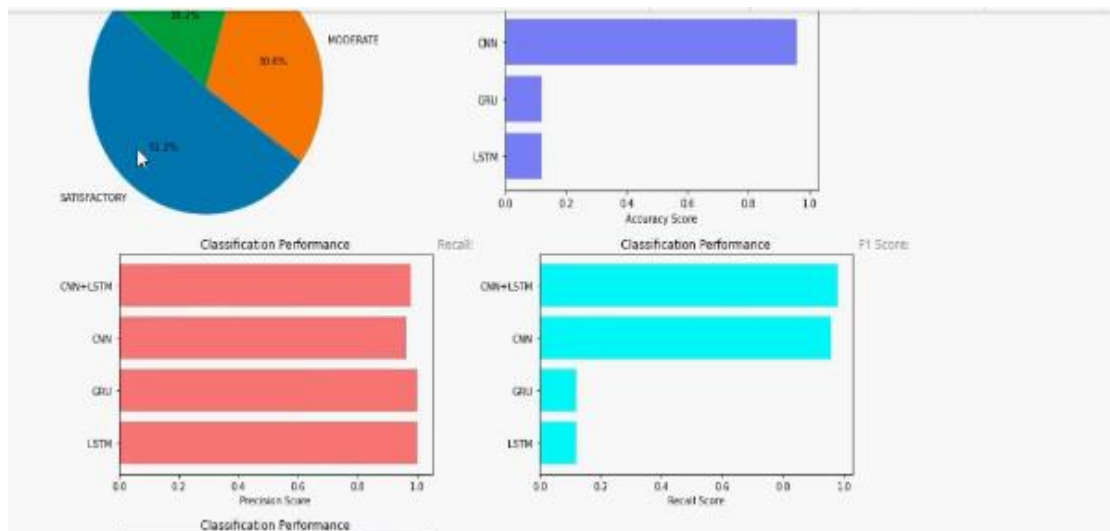


Fig.9.5 Charts

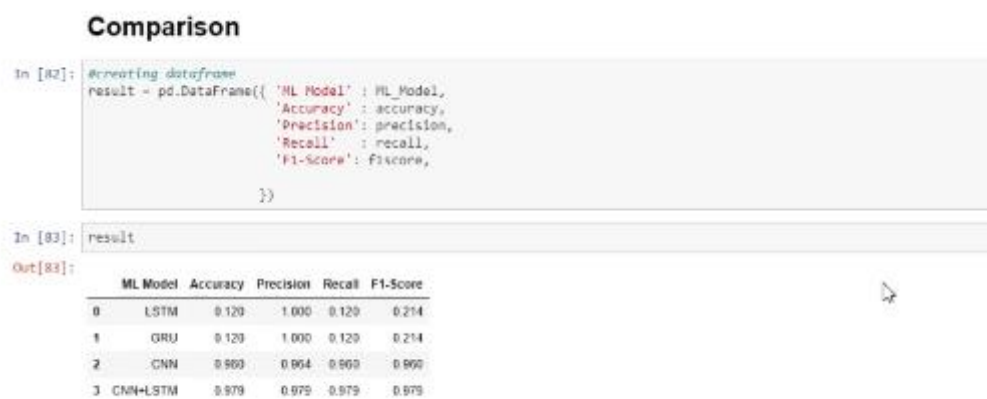


Fig.9.6 Comparison

Season

Spring

PM2.5

6.55

PM2.5 AQI


27

NO2

0.9

NO2 AQI

Fig.9.7 User Input

 AQI

HOMEABOUTNOTEBOOKSIGNOUT

Season

Summer

PM2.5

158.55

PM2.5 AQI

2

27

27.8

2.1238

206

23

Fig.9.8 User Input



Result: **Air Quality is Satisfactory!**

Fig.9.9 Output Page

10. CONCLUSION

We want to be able to predict most types of air pollution in one step, including NO₂, O₃, SO₂, PM_{2.5}, and PM₁₀. We will do this by using different predicting methods based on deep learning and statistical models. Evaluation measures like RMSE, MAE, and R² are used to test how well the predicting models work. Among all the forecasting models and pollution, the LSTM model had the lowest RMSE and MAE for predicting SO₂ time series data. On the other hand, the ARIMA model had the highest RMSE and MAE for modeling NO₂. In terms of R², both DL models did about the same as the other predictive models when it came to predicting O₃. For SO₂, on the other hand, the GRU model was found to be the least accurate. Overall, the results showed that out of all the predicting models that were looked at, DL models constantly did better than statistical models, getting the lowest error rates in terms of RMSE and MAE for all the pollutants and higher predictive accuracy rates in terms of R² for most of them. The ARIMA model was only better at identifying two pollutants (SO₂ and PM₁₀) based on the R² score. However, it had the best RMSE and MAE values for all of the pollutants. We want to improve the performance of deep learning models and make predictions that take more than one step in the future by using new feature engineering methods and optimizing the models' hyperparameters.

FUTURE SCOPE

Our goal for the future is to move toward multi-step prediction and improve the performance of deep learning (DL) models by coming up with new ways to build features and finetune the tuning of hyperparameters. By focusing on multi-step prediction, we hope to broaden the scope of predicting beyond single time steps, giving us a better picture of how much pollution will be in the air in the future. We also want to try out new ways of using feature engineering to get more useful information from the data. This will help the deep learning models better understand complex patterns and connections. In addition, fine-tuning model setups to get the best predicted accuracy and stability will be part of improving hyperparameters. These improvements will not only make DL models better at predicting air quality, but they will also help us learn more about how the world works and make better decisions about how to lessen the bad effects of air pollution.

11. REFERENCES

- [1] Brook, Robert D. "Cardiovascular effects of air pollution." *Clinical science* 115.6 (2008): 175-187.
- [2] Stafoggia, Massimo, and Tom Bellander. "Short-term effects of air pollutants on daily mortality in the Stockholm county—a spatiotemporal analysis." *Environmental Research* 188 (2020): 109854.
- [3] World Health Organization. "Household air pollution." *Date last updated* 28 (2022).. Available: <https://www.who.int/news-room/fact-sheets/detail/household-air-pollution-and-health>
- [4] WHO. Air Quality and Health. Accessed: Jan. 10, 2023.[Online]. Available: <https://www.who.int/teams/environment-climate-change-and-health/air-quality-and-health/policy-progress/sustainable-development-goals-air-pollution>
- [5] B. Paul and S. Louise. (2022). Air Quality: Policies, Proposals and Concerns—House of Commons Library. Accessed: Jan. 10, 2023.[Online]. Available: <https://commonslibrary.parliament.uk/researchbriefings/cbp-9600/>
- [6] Landrigan, Philip J. "Air pollution and health." *The Lancet Public Health* 2.1 (2017): e4-e5.
- [7] Abutalip, Kudaibergen, Anas Al-Lahham, and Abdulmotaleb El Saddik. "Digital twin of atmospheric environment: Sensory data fusion for high-resolution PM 2.5 estimation and action policies recommendation." *IEEE Access* 11 (2023): 14448-14457.
- [8] Ma, Jun, et al. "Identification of high impact factors of air quality on a national scale using big data and machine learning techniques." *Journal of Cleaner Production* 244 (2020): 118955.
- [9] Manisalidis, Ioannis, et al. "Environmental and health impacts of air pollution: a review." *Frontiers in public health* 8 (2020): 505570.
- [10] Ameer, Saba, et al. "Comparative analysis of machine learning techniques for predicting air quality in smart cities." *IEEE access* 7 (2019): 128325-128338.
- [11] Chen, Qi, et al. "A survey on an emerging area: Deep learning for smart city data." *IEEE Transactions on Emerging Topics in Computational Intelligence* 3.5 (2019): 392-410.
- [12] Zhang, Ying, et al. "A predictive data feature exploration-based air quality prediction approach." *IEEE Access* 7 (2019): 30732-30743.
- [13] Harishkumar, K. S., K. M. Yogesh, and Ibrahim Gad. "Forecasting air pollution particulate matter (PM_{2.5}) using machine learning regression models." *Procedia Computer Science* 171 (2020): 2057-2066..

[14] Chang, Yue-Shan, et al. "An LSTM-based aggregated model for air pollution forecasting." *Atmospheric Pollution Research* 11.8 (2020): 1451-1463.

[15] Ma, Jun, et al. "A Lag-FLSTM deep learning network based on Bayesian Optimization for multi-sequential-variant PM2. 5 prediction." *Sustainable Cities and Society* 60 (2020): 102237.