**ABSTRACT**

The degradation of air quality due to various anthropogenic and natural factors poses significant threats to human health and the environment. Monitoring and predicting air quality have become imperative tasks for mitigating these adverse effects. The Air Quality Index (AQI) serves as a crucial indicator, summarizing air quality data into a single numerical value for easy interpretation. Traditional methods of AQI estimation often rely on statistical models or physical simulations, which may have limitations in accuracy and efficiency. In recent years, machine learning (ML) techniques have emerged as powerful tools for AQI prediction due to their ability to handle complex, nonlinear relationships inherent in air quality data. This paper presents a comprehensive review of machine learning-based approaches for AQI prediction. Various ML algorithms such as support vector machines, random forests, neural networks, and ensemble methods have been applied to AQI prediction tasks with promising results. This review highlights the strengths and limitations of different ML algorithms in the context of AQI prediction, including data pre-processing, feature selection, and model evaluation techniques. Additionally, it discusses the challenges associated with real-time AQI prediction, such as data sparsity, sensor placement, and model interpretability. Furthermore, this paper discusses potential future directions for research in ML-based AQI prediction, including the integration of multi-source data (e.g., satellite imagery, meteorological data) for improved accuracy, the development of hybrid models combining physics-based and data-driven approaches, and the deployment of edge computing for real-time AQI monitoring in smart cities. In conclusion, machine learning offers significant potential for enhancing AQI prediction accuracy and facilitating proactive measures for air quality management. However, addressing the remaining challenges requires interdisciplinary collaboration among researchers, policymakers, and stakeholders to realize the full benefits of ML in improving air quality and public health.

**Chapter I**

**Introduction**

**CHAPTER 1**

**INTRODUCTION**

**1.1 Project Introduction**

Air pollution is a pressing environmental issue that affects the health and well-being of millions worldwide. The Air Quality Index (AQI) serves as a vital tool for communicating the quality of the air we breathe to the public. It aggregates data on various air pollutants into a single numerical value, providing an easy-to-understand indicator of air quality levels and associated health risks. Traditionally, AQI computation relies on mathematical models and empirical formulas based on pollutant concentrations. However, these methods often face challenges in accurately capturing the complex dynamics of air pollution, especially in urban areas with diverse emission sources and meteorological influences. Moreover, traditional approaches may struggle to adapt to rapidly changing conditions and lack the flexibility to incorporate diverse data sources. In recent years, machine learning (ML) has emerged as a powerful approach for AQI prediction and monitoring. ML techniques offer the ability to extract patterns and relationships from large and diverse datasets, enabling more accurate and timely AQI predictions. By leveraging the capabilities of ML algorithms, researchers can develop models that account for the nonlinear interactions between various pollutants, meteorological factors, and geographical features, leading to improved AQI estimation. This introduction sets the stage for exploring the application of machine learning in AQI prediction. It outlines the limitations of traditional methods and highlights the potential of ML to address these challenges. Furthermore, it underscores the importance of accurate AQI prediction in informing public health decisions, supporting environmental policies, and promoting community awareness and engagement. This paper delves into the current state-of-the-art in ML-based AQI prediction, discussing various algorithms, data sources, and model evaluation techniques. We also examine the challenges and opportunities associated with ML approaches, including data quality issues, model interpretability, and scalability for real-time applications. Finally, we outline future directions for research and development in this rapidly evolving field, emphasizing the potential for ML to transform air quality monitoring and management efforts for the benefit of society. This introduction sets the stage for exploring the application of machine learning in AQI prediction. Furthermore, it underscores the importance of accurate AQI prediction in informing public health decisions, supporting environmental policies, and promoting community awareness and engagement. Moreover, we discuss the societal implications of accurate AQI prediction, including its role in reducing health disparities, informing policy interventions, and fostering community resilience. By harnessing the power of ML, we can develop innovative solutions for air quality monitoring and management that enhance public health outcomes and promote sustainable development. Finally, we outline future directions for research and development in this rapidly evolving field, emphasizing the potential for ML to transform air quality monitoring and management efforts for the benefit of society. Through interdisciplinary collaboration and stakeholder engagement, we can leverage ML technologies to address the multifaceted challenges of air pollution and create healthier and more liveable environments for current and future generations.

**1.2 Scope of the Project**

The scope of an Air Quality Index (AQI) prediction project is broad and impactful, given the increasing global concern over air pollution and its detrimental effects on human health and the environment. Such a project involves developing predictive models that estimate AQI levels based on various environmental and meteorological factors such as particulate matter (PM2.5 and PM10), ozone (O3), nitrogen dioxide (NO2), sulfur dioxide (SO2), carbon monoxide (CO), temperature, humidity, wind speed, and geographical location. The primary objective is to provide accurate and timely forecasts of air quality, enabling individuals, communities, and policymakers to take proactive measures to mitigate pollution-related health risks. The scope encompasses data collection, pre-processing, feature engineering, model selection, training, validation, and deployment of machine learning or statistical models. Additionally, integrating real-time sensor data, satellite imagery, weather forecasts, and historical AQI data can enhance the predictive capabilities of the models. The scope also extends to collaboration with governmental agencies, research institutions, environmental organizations, and urban planners to leverage the predictions for policy formulation, urban development, and environmental conservation initiatives. Overall, an AQI prediction project has significant potential to improve public health, raise awareness about air quality issues, and promote sustainable environmental practices.

**1.3 Objective of the project**

The objective of an Air Quality Index (AQI) prediction project is to develop a reliable and accurate model that can forecast the AQI levels in a given area over a specific period. By predicting AQI levels, this project aims to provide valuable insights into air quality conditions, enabling individuals, communities, and policymakers to take proactive measures to protect public health and the environment. Key objectives of such a project include leveraging historical air quality data, meteorological data, and possibly other relevant factors to develop robust predictive models capable of accurately forecasting AQI levels. These models can then be used to generate real-time or future AQI predictions, aiding in the implementation of targeted interventions, such as traffic management, emission controls, and public health advisories, to mitigate the adverse effects of air pollution on human health and well-being. Ultimately, the goal is to improve air quality monitoring and management efforts, leading to healthier and more sustainable living environments for all.

**Chapter II**

**Software**

**Requirement Analysis**

**CHAPTER II**

# SYSTEM CONFIGUARTION

## 2.1 Hardware Requirements

|  |  |  |
| --- | --- | --- |
| • | Processor | : Intel core processor 2.6.0 GHZ |
| • | RAM | : 8GB |
| • | Hard disk | : 430 GB |
| • | Compact Disk | : 650 Mb |
| • | Keyboard | : Standard keyboard |
| • | Monitor | : 15inch colour monitor |

## 2.2 Software Requirements

* Operating system : Windows OS
* Programming language : PYTHON
* Platform : Google Colaboratory

## 2.3 About Software and its Description

### Python

Python is a high-level, versatile, and widely used programming language known for its simplicity and readability. It is commonly used in a variety of domains, including web development, data analysis, scientific computing, machine learning, artificial intelligence, and more. Python is one of the most popular programming languages for machine learning projects, and there are several compelling reasons for its widespread use in this field:

### 1. Extensive Libraries and Frameworks

Python has a rich ecosystem of machine learning libraries and frameworks, such as

TensorFlow, Keras, PyTorch, Scikit-learn, and many others. These libraries provide preimplemented algorithms, tools, and APIs that streamline the development of machine learning models.

### 2. Ease of Learning

Python is known for its simplicity and readability, which makes it an excellent choice for beginners in the field of machine learning. Its clean and concise syntax allows developers to focus on the machine learning concepts rather than getting bogged down in complex code.

### 3. Versatility

Python is a versatile language, meaning you can use it for various stages of a machine learning project, from data preprocessing and exploration to model development and deployment.

This versatility reduces the need to switch between languages or tools.

### 4. Strong Data Analysis and Visualization Capabilities

Python has powerful data analysis and visualization libraries, including pandas, NumPy, and Matplotlib. These tools are essential for tasks like data cleaning, feature engineering, and creating informative visualizations.

### 5. Open Source

Python is open-source, which means it's free to use and has a vast community contributing to its development. This reduces project costs and allows for customization.

### 6. Cross-Platform Compatibility

Python is a cross-platform language, so machine learning projects developed on one operating system can easily be transferred and run on others, such as Windows, MacOS, or Linux.

**7. Scalability**

Python can be used for small-scale projects, and it's also suitable for large-scale, production-ready machine learning systems when combined with frameworks like TensorFlow and PyTorch.

### 8. Community Packages

Python's extensive package ecosystem goes beyond machine learning. You can easily integrate machine learning components with web development, databases, and other domains, expanding the capabilities of your project.

## 2.4 Google Colab

Google Colab, short for Google Colaboratory, is a popular cloud-based platform for running Python projects, especially in the fields of data science and machine learning. There are several reasons why you might consider using Google Colab for your Python project:

### 1. Free Cloud Computing

Google Colab provides free access to cloud computing resources. It offers a free GPU (Graphics Processing Unit) for running deep learning and data-intensive tasks, which can be expensive to set up and maintain on your local machine.

### 2. No Setup Required

You don't need to set up your development environment. It comes pre-installed with many commonly used data science libraries such as NumPy, Pandas, Matplotlib, and TensorFlow, making it easy to start working on your project without spending time configuring your local environment.

### 3. Collaboration

Google Colab is built for collaboration. You can easily share your Colab notebooks with others, and multiple users can work on the same notebook simultaneously, making it a great tool for team projects.

### 4. Integration with Google Drive

You can save your Colab notebooks in Google Drive and access them from anywhere. This seamless integration with Google Drive also allows you to store datasets and other project-related files.

### 5. Access to Google Services

You can make use of other Google services such as Big Query for data analysis, Google Sheets for data storage, and more, directly within your Colab notebook.

### 6. Easy to Share

Sharing your work is simplified through shareable links. You can publish your notebook or share it privately, making it an excellent tool for education and data science competitions.

### 7. Interactive Documentation

Colab notebooks are a great way to create interactive documentation for your Python project. You can mix code, visualizations, and explanatory text in the same document.

### 8. Hardware Acceleration

As mentioned earlier, Colab provides free access to GPUs, which is crucial for training deep learning models that require significant computational power.

### 9. Flexibility

You're not limited to just Python; you can run code in other languages and even shell commands within a Colab notebook.

**Libraries Used**

### Tools Employed

* NumPy
* Pandas
* Matplotlib
* Pickle
* Sklearn
* SciPy
* Seaborn
* ydata\_profiling

### NumPy

* NumPy is a Python library for scientific computing.
* It provides a high-performance implementation of multidimensional arrays and matrices, as well as a large collection of mathematical functions for operating on those arrays.
* NumPy is widely used in data science, machine learning, and scientific computing.

### Features

* Multidimensional arrays: NumPy provides a high-performance implementation of multidimensional arrays, which are essential for many scientific and engineering applications.
* Mathematical functions: NumPy provides a large collection of mathematical functions for operating on multidimensional arrays, including linear algebra, statistical, and Fourier transform functions.
* Broadcasting: NumPy provides a powerful broadcasting mechanism that allows for efficient operations on arrays of different shapes.

### Pandas

* Pandas is a Python library for data analysis.
* It is built on top of the NumPy library and provides high-level data structures and operations for manipulating numerical data and time series.
* Pandas is widely used in data science, machine learning, and financial analysis.

### Features

* Data structures: Pandas provides two main data structures for storing and manipulating data: Series and DataFrame. A Series is a one-dimensional labelled array, similar to a Python list. A DataFrame is a two-dimensional labelled array, similar to a Python dictionary.
* Data manipulation: Pandas provides a wide range of functions for manipulating data, including sorting, filtering, grouping, and aggregating.
* Data visualization: Pandas integrates with Matplotlib, a Python library for data visualization, to provide easy-to-use plotting functions.
* Time series analysis: Pandas provides a variety of tools for working with time series data, such as date parsing, time shifting, and resampling.

### Matplotlib

* Matplotlib is a Python library for data visualization.
* It provides a wide range of plotting functions for creating static, animated, and interactive visualizations.
* Matplotlib is widely used in data science, machine learning, and scientific computing.

### Features

* Comprehensive set of plotting functions: Matplotlib provides a wide range of plotting functions for creating line plots, bar plots, scatter plots, histograms, and many other types of visualizations.
* Easy to use: Matplotlib has a simple and intuitive interface, making it easy to learn and use.
* Flexible and customizable: Matplotlib is highly customizable, allowing users to create visualizations that meet their specific needs.
* Well-documented: Matplotlib is well-documented, with a comprehensive user guide and tutorials.

### Scikit learn

* Scikit-learn is a free software machine learning library for the Python programming language.
* It features various classification, regression, clustering and dimensionality reduction algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
* Scikit-learn is widely used in data science, machine learning, and natural language processing.

### Features

* Comprehensive set of algorithms: Scikit-learn provides a wide range of machine learning algorithms for classification, regression, clustering, and dimensionality reduction.
* Easy to use: Scikit-learn has a consistent and user-friendly interface, making it easy to learn and use.
* Efficient and scalable: Scikit-learn is written in a performance-critical fashion, making it efficient and scalable for large datasets.

### Pickle

The pickle library is a built-in Python module that provides a mechanism for serializing and deserializing Python object structures. In other words, it allows Python objects to be converted into a byte stream that can be stored or transmitted, and then later reconstructed back into the original object.

### Features

* Serialization of most Python objects: The pickle library can serialize almost any Python object, including built-in types, user-defined classes, and instances of those classes.
* Recursive object support: The pickle library can handle recursive objects, which are objects that contain references to themselves.
* Object sharing: The pickle library can share objects between different serialization and deserialization operations. This means that if an object is pickled multiple times, only a single copy of the object will be stored.
* Portability: The pickle library is portable between different versions of Python. This means that a pickled object can be deserialized using a different version of Python than the one used to pickle it.

**Seaborn**

Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

**Features**

* Seaborn is a Python data visualization library built on top of Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Some of the key features of Seaborn include:
* Seaborn provides a simple and intuitive interface for creating complex statistical visualizations with minimal code.
* Seaborn comes with built-in themes that improve the aesthetics of plots. Themes include dark grid, white grid, dark, white, and ticks, among others.
* Seaborn offers a variety of color palettes for enhancing the visual appeal of plots. These palettes are designed to work well with the human perceptual system.

**ydata**\_**profiling**

With a focus on providing a one-line Exploratory Data Analysis (EDA) solution, ydata Profiling delivers an extended analysis of your Data Frame, akin to the convenience offered by pandas' df. Describe () function. This analysis can be seamlessly exported in different formats such as HTML and JSON.

**Features**

* ydata Profiling is a process of analyzing and understanding data in order to gain insights into its structure, quality, and characteristics. The features of ydata Profiling typically include:
* ydata Profiling provides a summary of the dataset, including the number of rows and columns, data types, and basic statistics such as mean, median, mode, minimum, maximum, and standard deviation for numerical columns.
* ydata Profiling assesses the quality of the data by identifying missing values, duplicate records, outliers, and inconsistencies.
* It visualizes the distribution of values within each column using histograms, density plots, box plots, and other graphical representations.

**Dataset Description**

**Title**

Comparative Analysis Using Machine learning on Air quality index prediction

## Data Source

The dataset contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multi-sensor Device. The device was located on the field in a significantly polluted area, at road level, within an Italian city. Data were recorded from March 2004 to February 2005 (one year) representing the longest freely available recordings of on-field deployed air quality chemical sensor device responses. Ground Truth hourly averaged concentrations for CO, Non-Metanic Hydrocarbons, Benzene, Total Nitrogen Oxides (NOx), and Nitrogen Dioxide (NO2) were provided by a co-located reference certified analyzer. Evidences of cross-sensitivities as well as both concept and sensor drifts are present as described in De Vito et al., Sens. And Act. B, Vol. 129,2,2008 (citation required) eventually affecting sensors concentration estimation capabilities. Missing values are tagged with -200 value. This dataset can be used exclusively for research purposes. Commercial purposes are fully excluded.

**Attribute Information:**

Date (DD/MM/YYYY).

Time (HH.MM.SS).

True hourly averaged concentration CO in mg/m^3 (reference analyzer)

PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted) True hourly averaged overall Non-Metanic Hydro Carbons concentration in microg/m^3 (reference analyzer).

True hourly averaged Benzene concentration in microg/m^3 (reference analyzer).

PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted) True hourly averaged NOx concentration in ppb (reference analyzer).

PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted) True hourly averaged NO2 concentration in microg/m^3 (reference analyzer).

PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted).

PT08.S5 (indium oxide) hourly averaged sensor response (nominally O3 targeted).

Temperature in Â°C.

Relative Humidity (%).

AH Absolute Humidity.

**Chapter III**

**System Analysis**

**CHAPTER III**

# SYSTEM ANALYSIS

**3.1 Existing System**

The existing system for the Air Quality Index (AQI) typically involves a network of monitoring stations strategically placed in urban and industrial areas to measure various pollutants in the air. These sub-indices are then combined to calculate the overall AQI value, which is typically reported on a scale from 0 to 500 or higher. The AQI value is categorized into different levels (e.g., good, moderate, unhealthy, very unhealthy, hazardous) to provide users with information about the current air quality conditions and associated health risks. The existing system also includes dissemination mechanisms for communicating AQI information to the public through various channels such as websites, mobile apps, social media, and traditional media outlets. This information allows individuals, communities, and policymakers to make informed decisions to protect public health and the environment, such as limiting outdoor activities during times of poor air quality, implementing emission reduction measures, and issuing public health advisories. The existing system for AQI plays a critical role in monitoring and managing air quality, providing valuable information to stakeholders for taking proactive measures to improve air quality and safeguard public health.

**Disadvantages of the Existing system:**

* Despite efforts to disseminate AQI information through various channels, including websites, mobile apps, and traditional media, reaching all segments of the population can be challenging. Language barriers, limited internet access, and disparities in digital literacy may hinder the effectiveness of communication efforts.
* In some cases, AQI data may not be available in real-time, causing delays in informing the public about rapidly changing air quality conditions and limiting the ability to take timely preventive actions.

## 3.2 Proposed system

The proposed system for Air Quality Index (AQI) aims to offer a comprehensive solution for monitoring and predicting air quality levels in a given region. This system integrates various components to collect, process, analyze, and visualize data related to air quality parameters. At its core, the system includes sensors deployed across the region to measure key pollutants such as nitrogen dioxide (NO2), sulfur dioxide (SO2), carbon monoxide (CO), and ozone (O3). These sensors continuously gather data on pollutant levels, which is then transmitted to a central database in real-time. The collected data is processed and analyzed using advanced algorithms and statistical techniques to calculate the Air Quality Index for the region. This index provides a standardized measure of overall air quality, making it easier for individuals and authorities to interpret and act upon. In addition to real-time monitoring, the system incorporates predictive modeling capabilities to forecast future AQI levels based on historical data, meteorological factors, and other relevant parameters. To make the information accessible and actionable, the system includes user-friendly interfaces such as web dashboards and mobile applications. These interfaces provide real-time AQI updates, historical trends, interactive maps, and personalized alerts to inform and empower users to make informed choices regarding outdoor activities, health precautions, and environmental policies. Furthermore, the proposed system emphasizes scalability, adaptability, and interoperability, allowing for seamless integration with existing air quality monitoring networks, government agencies, research institutions, and public health organizations. This collaborative approach fosters data sharing, transparency, and collective efforts towards addressing air pollution challenges and promoting environmental sustainability.

**Advantages of the Proposed system:**

* The system collects data from multiple sensors measuring various pollutants, providing a comprehensive view of air quality conditions in the region.
* The system calculates the AQI using standardized methods, facilitating easy interpretation and comparison of air quality levels across different regions and periods.
* User-friendly interfaces such as web dashboards and mobile applications make AQI information accessible to a wide range of users, including the general public, policymakers, and researchers.

**Chapter IV**

**System Design**

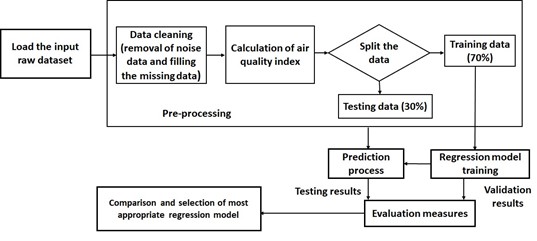
**CHAPTER IV**

# SYSTEM DESIGN

## 4.1 System Architecture

The System Architecture consists of 6 phases: The six primary parts of this approach are as follows: splitting and balancing data, AQI computation, feature selection from data, and dataset pre-processing. In the first step, datasets on air quality were gathered and loaded for analysis. The next action following data collection is to guarantee data quality, pre-processing techniques were used, such as addressing missing values and minimizing outliers. The AQI Calculation stages for the air contaminants in the dataset are then applied. Following data processing, the most pertinent and crucial data are extracted using the feature extraction technique. This stage concentrates primarily on the important variables while assisting in lowering the dataset's null and missing values. The dataset was then separated into training and testing sets after being processed to provide an equal representation of the various classes. In the end, our regression models create regression classifiers to forecast air quality by using the dataset and its key features as input. To find an appropriate and effective model for forecasting the air quality index, performance measures were examined and calculated.

## 4.2 Workflow



1. Dataset Description:

Datasets can be sourced from various sources, including experiments, surveys, and observations, or generated synthetically for specific purposes. They can also be publicly available or proprietary, depending on the context in which they are used. We use “Kaggle” as our dataset store. In which we use the Air Quality Index Dataset of Italian City recorded between March 2004 to February 2005. An example of this problem with a dataset of 9358 with five metal oxide chemical sensors stacked in an AQI chemical detection device is included in the dataset. To detect chemicals, the truth hourly averaged amounts of CO, NOx, NO2, benzene, and total nitrogen oxides (NOx) are evaluated for one year. The dataset's missing values are designated with a -200 value. Date, Time, PT08.S1 (tin oxide), PT08.S2 (titania), PT08.S3 (tungsten oxide), PT08.S4 (tungsten oxide), PT08.S5 (indium oxide), Temperature in Â°C, RH (relative humidity) (%), and AH (absolute humidity) are the primary attribute information for the dataset.

1. Dataset Pre-processing:

Data pre-processing is an iterative process that often requires experimentation and domain knowledge to determine the most effective techniques for a given dataset and modeling task. Proper pre-processing can significantly improve the performance and reliability of data analysis and machine learning models. This process starts with handling the missing values using Exploratory Data Analysis (EDA) which is a critical step in understanding and preparing data for pre-processing. EDA helps in identifying patterns, trends, anomalies, and relationships within the dataset, which in turn guides the pre-processing steps. We can perform EDA for data pre-processing by computing the basics of statistics such as Arithmetic Mean, minimum, maximum, and quantiles for numerical features. The presence and distribution of missing values in the dataset. Determine the appropriate strategy for handling missing data, such as imputation or removal, based on the nature and extent of missingness. Then, creating visualizations such as histograms, box plots, scatter plots, and bar charts to explore the distribution and relationships among variables.

1. AQI Computation:

As we know AQI is an important parameter for measuring Air Quality. It helps us provide a standard measure and its effects on the environment and human health. The Air Quality Index (AQI) is a numerical scale used to communicate how polluted the air currently is or how polluted it is forecasted to become. It is often used by government agencies to inform the public about local air quality conditions and associated health risks. The concentration levels of particular air pollutants, such as particulate matter like NMHC, NOx, NO2, ozone (O3), sulfur dioxide (SO2), nitrogen dioxide (NO2), and carbon monoxide (CO), are used to construct the Air Quality Index (AQI).

1. Feature Selection:

In our research, feature selection becomes important after the steps of data preparation and exploratory data analysis. In order to depict the overall state of air quality, the process entails locating and choosing the most pertinent features associated with the Air quality index. This study's characteristics are derived from a pre-processed AQI dataset that includes the AQI values for several pollutant components, including CO, NMHC, NOx, NO2, and O3. The association between the characteristics and AQI was discovered through the application of correlation analysis. The linear link between two variables can be ascertained by correlation analysis. The correlated values are compiled together into a correlation matrix which views the relationship between all values and variables. Then, we can evaluate each feature's predictive power in comprehending and forecasting variations in air pollution levels by computing the correlation coefficients between it and the AQI.

1. Splitting Data:

Splitting data is an essential phase in the creation of machine learning models. To do this, the accessible dataset is divided into several subsets for testing, validation, and training. It's essential to ensure that the data-splitting process maintains the original distribution of classes or labels, especially for classification tasks. This helps prevent biases in model evaluation and ensures that the trained model generalizes well to unseen data. Additionally, it's crucial to perform data splitting randomly to avoid introducing biases into the subsets. Random shuffling of the dataset before splitting helps ensure that each subset contains a representative sample of the data. In Python, libraries like sci-kit-learn provide convenient functions for splitting datasets into training, validation, and test sets. For example, the `train test split` function can be utilized to divide the data into sets for training and testing, and if desired, to future divide the training data into sets for training and validation.

1. Balancing Data:

Resolving unbalanced data issues, such as missing data and null values, is an essential step in machine learning activities to guarantee accurate and dependable prediction outcomes. According to this study's AQI value distribution, there is an imbalance in the AQI dataset, with some values occurring more frequently than others. By segmenting the AQI readings into predetermined ranges, this may be shown. Regression algorithms can be considerably improved by using unbalanced data. Next, more representative and multiple sets of data points are used to train the regression models by balancing the dataset. This step will improve the model's capacity to examine trends and connections among various air pollution AQI values, resulting in a well-developed model and more precise results.

### 4.3 Implementation Process

Implementing Air Quality Index project involves several steps, from data preparation and model development to deployment and evaluation. The implementation process of Air Quality Index (AQI) prediction using machine learning typically involves several key steps:

Data Collection:

Gather historical air quality data from reliable sources such as government agencies, research institutions, or environmental monitoring networks. This data should include measurements of pollutants such as NHMC, NO2, SO2, CO, and O3, as well as meteorological variables like temperature, humidity, wind speed, and precipitation.

Data Pre-processing:

Clean and pre-process the collected data to handle missing values, outliers, and inconsistencies. This may involve techniques such as imputation, outlier detection, and data normalization or scaling to ensure uniformity and quality.

Feature Engineering:

Extract relevant features from the pre-processed data that can serve as inputs to the machine learning models. This may include creating lag features, aggregating data over time intervals, or incorporating domain knowledge to enhance the predictive power of the model.

Model Selection:

Choose appropriate machine learning algorithms for AQI prediction, considering factors such as the nature of the data, the complexity of relationships between variables, and computational efficiency. Commonly used algorithms for time series forecasting tasks include linear regression, decision trees, random forests, support vector machines, and neural networks.

Model Training:

Split the pre-processed data into training and validation sets. Train the selected machine learning models on the training data, tuning hyperparameters as needed to optimize performance. Use the validation set to evaluate the models and select the best-performing one based on predefined evaluation metrics such as mean absolute error (MAE), root mean square error (RMSE), or coefficient of determination (R-squared).

Model Evaluation:

Assess the performance of the trained model on unseen data using appropriate evaluation metrics. Validate the model's ability to generalize to new observations and detect any potential issues such as overfitting or underfitting.

Model Deployment:

Once a satisfactory model is obtained, deploy it in a production environment for real-time AQI prediction. This may involve integrating the model into existing software systems, developing APIs for data input and output, and implementing mechanisms for model monitoring and maintenance.

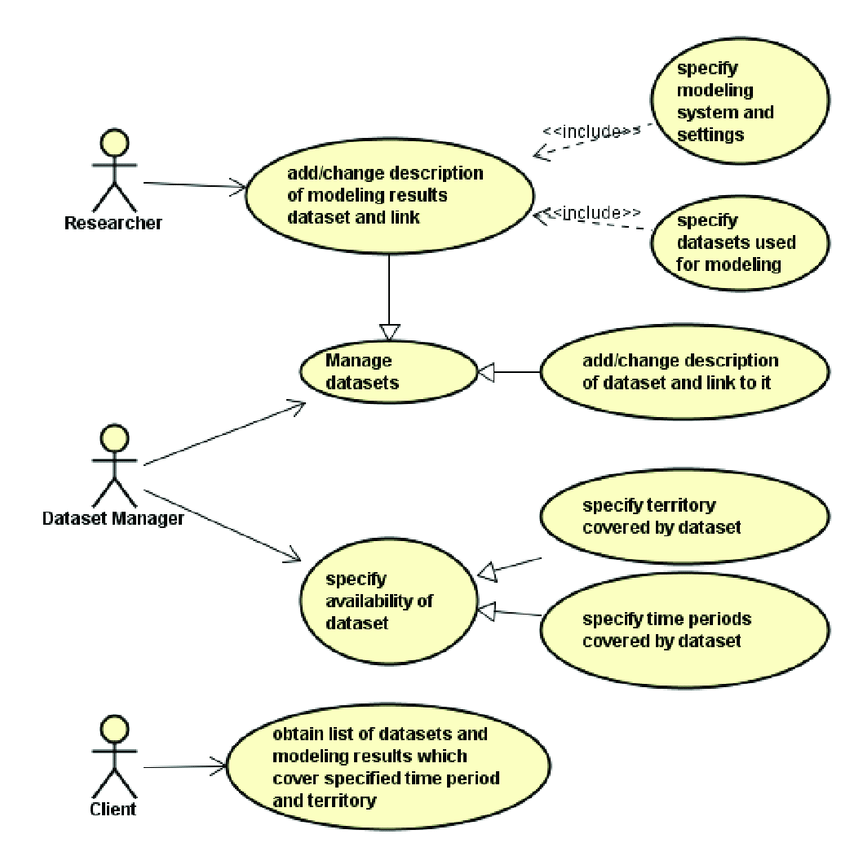
Continuous Improvement:

Monitor the performance of the deployed model over time and incorporate feedback to continuously improve its accuracy and reliability. This may involve retraining the model periodically with updated data, refining feature engineering techniques, or experimenting with different machine learning algorithms to achieve better results.

An effective AQI prediction system can be developed using machine learning techniques, helping to provide valuable insights into air quality conditions and support informed decision-making for environmental management and public health protection.

### 4.4 Use-case Diagram

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using pollutants and use cases. Use cases are a set of actions, services, and functions that the system needs to perform.



**Chapter V**

**System Implementation**

**CHAPTER V**

# MODEL IMPLEMENTATION

## 5.1 Algorithms used

Several machine learning algorithms can be used for Air Quality Index (AQI) prediction, depending on the specific requirements of the problem and the characteristics of the data. Some commonly used ML algorithms for AQI prediction include:

Linear Regression:

Linear regression is a simple yet powerful algorithm used for predicting a continuous target variable (in this case, AQI) based on one or more input features (such as pollutant concentrations, and meteorological data). It's particularly suitable when there is a linear relationship between the input features and the target variable.

Random Forest:

Random Forest is an ensemble learning algorithm that builds multiple decision trees and combines their predictions to improve accuracy and robustness. It is well-suited for AQI prediction as it can handle large datasets with high dimensionality and capture complex interactions between pollutants and meteorological factors.

Gradient Boosting Machines (GBM):

GBM is another ensemble learning technique that builds a sequence of weak learners (usually decision trees) in a step-wise manner, with each new learner focusing on the residual errors of the previous ones. GBM is known for its high predictive accuracy and ability to handle noisy data, making it suitable for AQI prediction tasks.

Support Vector Machines (SVM):

## SVM is a supervised learning algorithm that can be used for regression tasks as well. It works by finding the hyperplane that best separates the data points into different classes or predicts the target variable. SVMs are effective in capturing complex relationships in high-dimensional spaces and can be used for AQI prediction when there is a need for nonlinear modeling.

MLP Classifier:

The Multilayer Perceptron (MLP) classifier is a type of artificial neural network (ANN) that is commonly used for classification tasks. It belongs to the family of feedforward neural networks and consists of multiple layers of nodes (neurons), including an input layer, one or more hidden layers, and an output layer. MLP classifiers are flexible and capable of learning complex decision boundaries, making them suitable for a wide range of classification tasks. However, they require careful hyperparameter tuning and may be prone to overfitting, especially when dealing with high-dimensional data or small datasets. Regularization techniques, such as dropout and L2 regularization, can help mitigate overfitting in MLP classifiers.

K-Nearest Neighbors (KNN):

KNN is a simple yet effective algorithm that predicts the target variable based on the majority vote or average of its nearest neighbors in the feature space. KNN is suitable for AQI prediction when there are spatial dependencies in the data, such as geographical proximity affecting pollutant concentrations.

AdaBoost:

AdaBoost, short for Adaptive Boosting, is an ensemble learning technique used primarily for classification tasks. It is a powerful algorithm that combines multiple weak classifiers to create a strong classifier. AdaBoost is known for its ability to improve the performance of weak learners by focusing on difficult instances and emphasizing their importance during training. It is also resistant to overfitting, as long as the weak classifiers are not too complex.

These are just a few examples of the machine learning algorithms that can be used for AQI prediction. The choice of algorithm depends on factors such as the nature of the data, the complexity of relationships between variables, computational resources, and the desired level of interpretability and accuracy. It's often beneficial to experiment with multiple algorithms and evaluate their performance using appropriate metrics before selecting the most suitable one for a given AQI prediction task.

**Chapter VI**

**System Testing**

**CHAPTER VI**

# SYSTEM TESTING AND IMPLEMENTATION

## 6.1 System Testing

Testing is carried out after the development of the proposed system. The principal activity of system development is preparing the source code. In this system the source code is developed for each module separately. The source code is prepared for master files and they are compiled and corrected. Then the source code for the transaction files is prepared, compiled and corrected. Then the modules are combined and corrected as a whole module.

A strategy for software testing must accommodate low-level tests that are necessary to verify that all small source code segments has been correctly implemented as well as high-level tests that validate major system functions against customer requirements. Testing is a process of executing program with the intent of finding error. A good test case is one that has high probability of finding an undiscovered error. If testing is conducted successfully, it uncovers the errors in the software. Testing cannot show the absence of defects, it can only show that software defects present. Test configuration includes test plan and test cases and test tools.

### Testing Objectives

Software Testing has different goals and objectives. The major objectives of Software testing are as follows:

* Finding defects which may get created by the programmer while developing the software.
* Gaining confidence in and providing information about the level of quality and to prevent defects.
* To make sure that the end result meets the business and user requirements.
* To ensure that it satisfies the BRS that is Business Requirement Specification and
* SRS that is System Requirement Specifications.
* To gain the confidence of the customers by providing them a quality product.

### Testing Methodologies

Testing methodologies are the strategies and approaches used to test a particular product to ensure it is fit for purpose. Testing methodologies usually involve testing that the product works in accordance with its specification, has no undesirable side effects when used in ways outside of its design parameters and worst case will fail-safely.

### • Unit Testing

Unit testing is essential for the verification of the code produced during the coding phase and hence the goal is to test the internal logic of the modules. Using the detailed design description as a guide, important paths are tested to uncover errors within the boundary of the modules. These tests were carried out during the programming stage itself.

### • Integration Testing

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover error associated with the interface. The objective is to take unit-tested modules and build a program structure that has been dictated by design. All modules are combined in this step. The entire program is tested as a whole. Chaos in interfaces may usually result. A set of errors is encountered in such a case.

### • Validation Testing

Here in the validation testing, we want to check whether the given conditions to the text box are working correctly. Because in the name place we want to enter the characters and the special symbols only we should not enter the numbers in the name field. Here while on runtime we entered numeric values in the string-specified columns of the product inwards. It raises errors. In this phase each module has been tested with wrong inputs, for example, Employee's Name should be a character as well as their age should be in numbers.

### • Functional Testing

The functional testing part of a testing methodology is typically broken down into four components - unit testing, integration testing, system testing, and acceptance testing – usually executed in this order. Entire system that is working properly or not will be tested here, and specified path connection is correct or not, and giving output or not are tested here these verifications and validations are done by giving input values to the system and by comparing with expected output.

## 6.2 System Implementation

Implementation is the stage in the project where the theoretical design is turned into a working system and gives confidence on the new system for the users that it will work efficiently and effectively. It involves careful planning, investigation of the current system and its constraints on implementation, design of methods to achieve the changeover, an evaluation of changeover methods. Apart from planning major tasks of preparing the implementation are education and training of users. The implementation process begins with preparing a plan for the implementation of the system.

According to this plan, the activities are to be carried out, discussions made regarding the equipment and resources and the additional equipment has to be acquired to implement the new system. In the network backup system no additional resources are needed. Implementation is the final and the most important phase. The most critical stage in achieving a successful new system is giving the users confidence that the new system will work and be effective. The system can be implemented only after thorough testing is done and if it is found to be working according to the specifications. This method also offers the greatest security since the old system can take over if errors are found or inability to handle certain types of transactions while using the new system.

**Chapter VII**

**Evaluation Metrics**

**CHAPTER VII**

# EVALUATION

## 7.1 Evaluation Metrics

Evaluation metrics for regression models are used to assess the performance of the model in predicting continuous numerical values. Here are some widely used evaluation metrics for regression models:

1. Mean Absolute Error (MAE):

The average of the absolute discrepancies between the values that were predicted and those that were observed is known as the mean absolute error. Without taking into account the direction of the errors, it gives us an estimate of their average size.

2. Mean Squared Error (MSE):

The average of the squared discrepancies between the expected and actual values is known as the mean squared error. Compared to MAE, it penalizes greater errors more severely.

3. Root Mean Squared Error (RMSE):

The square root of the MSE is the root mean squared error. It gives an understandable measure of the average error and is in the same unit as the goal variable.

4. Mean Absolute Percentage Error (MAPE):

The average of the absolute percentage disparities between the actual and anticipated values is known as the mean absolute percentage error. It gauges how accurate the model's predictions are in comparison.

5. R-squared (R2):

The percentage of the dependent variable's variation that can be predicted from the independent variables is expressed as the coefficient of determination (R-squared). Higher numbers indicate a better match between the model and the data. Its range is 0 to 1.

6. Adjusted R-squared:

An alternative form of R-squared that penalizes the addition of pointless predictors to the model is called adjusted R-squared. It offers a more precise indicator of the model's fit and accounts for the quantity of predictors in the model. When comparing models with varying amounts of predictors, it is especially helpful.

MAE = RMSE = =

MSE = = 1 -

Where, - Predicted value of y, - Mean value of y.

Several viewpoints on the regression model's performance are offered by these assessment metrics. To have a thorough grasp of the model's performance, it is important to take into account several metrics and select the most suitable one according to the task's particular needs. Our preliminary evaluation results demonstrate the effectiveness of our method in forecasting air quality by contrasting the predicted values produced by models with the actual values. Next, we compare these two sets of numbers graphically, which allows us to rapidly determine how close they are to one another and provides important information about the accuracy of each model. The real values that we plotted against the anticipated values obtained by applying a particular focus to the regression model results. The degree of alignment between the data points and the ideal regression line, represented by the blue line, determines the correctness of the model. The data points are clearly clustered at the bottom when looking at the linear regression findings.

**Chapter VIII**

**Conclusion**

**and**

**Future Enhancements**

**CHAPTER VIII**

# CONCLUSION AND FUTURE ENHANCEMENTS

## Conclusion

In conclusion, our experiment demonstrated the effectiveness of employing machine learning, specifically a Gradient Boosting regression model, for predicting air quality index (AQI) values based on hourly measurements of various air pollutants. With a mean squared error (MSE) of 0.89, our model showed promising performance in accurately estimating AQI levels, highlighting its potential for real-time air quality monitoring and public health management. Moreover, the feature importance analysis underscored the significant influence of particulate matter NO2 and ozone (O3) concentrations on AQI predictions, emphasizing the importance of targeting these pollutants in air quality management efforts. Overall, our study contributes valuable insights into leveraging machine learning techniques for enhancing air quality assessment and informing policy interventions to control the harmful impacts of air pollution on people and the environment.

**Future Enhancement**

To future enhance Air Quality Index (AQI) prediction, several approaches can be explored to improve the accuracy and reliability of the models. One avenue for enhancement involves integrating additional data sources and features into the prediction models. This could include incorporating data from satellite imagery, remote sensing technologies, or IoT devices to capture more comprehensive information about air quality factors such as atmospheric conditions, land use patterns, traffic emissions, and industrial activities. By leveraging a broader range of data sources, the prediction models can capture more nuanced relationships and dependencies, leading to more accurate AQI forecasts. Future advancements in machine learning algorithms and techniques offer opportunities for enhancement. Deep learning models, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), have shown promise in capturing complex spatial and temporal patterns in air quality data. By leveraging the capabilities of deep learning architectures, AQI prediction models can potentially achieve higher levels of accuracy and generalization, particularly in dynamic and heterogeneous environments. In summary, future enhancement of AQI prediction involves leveraging additional data sources, advancing machine learning techniques, incorporating domain expertise, and fostering continuous model refinement through real-time monitoring and feedback mechanisms. By embracing a multidisciplinary approach and harnessing the power of data-driven methodologies, AQI prediction models can better support efforts to monitor, manage, and mitigate the impacts of air pollution on public health and the environment.

**Chapter IX**

**Appendix**

**CHAPTER IX**

**APPENDIX**

## 9.1 Source Code

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib.widgets import Slider, Button

from tqdm import \*

from datetime import datetime

from ydata\_profiling import ProfileReport

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

from sklearn.linear\_model import Ridge, Lasso, ElasticNet

from sklearn.linear\_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor

from sklearn.gaussian\_process import GaussianProcessRegressor

from sklearn.gaussian\_process.kernels import DotProduct, WhiteKernel , RBF

from sklearn.neighbors import KNeighborsRegressor

from sklearn.neural\_network import MLPRegressor

from sklearn.svm import SVR

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

from sklearn.model\_selection import cross\_val\_predict

import pickle

import scipy.stats as stats

# For warnings

import warnings

warnings.filterwarnings("ignore")

#loading the data from csv to pandas dataframe

df=pd.read\_csv('/content/drive/MyDrive/AQI GFG/AirQualityUCI.csv',sep = ";", decimal = ",")

df.head()

df.drop(['Unnamed: 15','Unnamed: 16'],axis=1,inplace= True, errors='ignore')

df.head()

df.shape

df.info()

df.describe().T

# Replacing bad sensor readings designated by an entry of -200 with NaN

df.replace(to\_replace = -200, value = np.nan, inplace = True)

df.info()

df.describe().T

df.isnull().sum()

round( 100\*( df.isnull().sum() / len(df.index) ), 2 ).sort\_values(ascending = False)

df.drop('NMHC(GT)', axis = 1, inplace = True, errors = 'ignore')

df.head()

df = df.dropna()

df.shape

df['DateTime'] = df['Date'] + ' ' + df['Hour']

df.DateTime = df.DateTime.apply(lambda x: datetime.datetime.strptime(x, '%d/%m/%Y %H.%M.%S'))

df['Weekday'] = df['DateTime'].dt.day\_name()

df['Month'] = df['DateTime'].dt.month\_name()

df['Hour'] = df['DateTime'].dt.hour

df['Date'] = pd.to\_datetime(df['Date'], format = '%d/%m/%Y')

df.drop('Time', axis = 1, inplace = True, errors = 'ignore')

df = df[['Date', 'Month', 'Weekday', 'Hour', 'DateTime', 'CO(GT)','PT08.S1(CO)', 'C6H6(GT)', 'PT08.S2(NMHC)', 'NOx(GT)',

'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)', 'PT08.S5(O3)', 'T', 'RH', 'AH']]

df.head()

for i in df.columns:

print("Count of unique values in \033[1m{}\033[0m column are \033[1m{}\033[0m.".format(i, df[i].nunique()))

for i in df.columns:

print("Minimum and Maximum value in \033[1m{}\033[0m column are \033[1m{}\033[0m and \033[1m{}\033[0m respectively.".format(i, df[i].min(), df[i].max()))

l = ['Date', 'Hour', 'DateTime', 'CO(GT)', 'PT08.S1(CO)', 'C6H6(GT)', 'PT08.S2(NMHC)', 'NOx(GT)', 'PT08.S3(NOx)', 'NO2(GT)',

'PT08.S4(NO2)', 'PT08.S5(O3)', 'T', 'RH', 'AH']

for i in l:

q1 = df[i].quantile(0.25)

q3 = df[i].quantile(0.75)

iqr = q3 - q1

upper = q3 + 1.5 \* iqr

lower = q1 - 1.5 \* iqr

outliner\_df = df.loc[(df[i] < lower) | (df[i] > upper)]

if df.shape[0] != 0:

percentage\_outliers = (outliner\_df.shape[0] / df.shape[0]) \* 100

else:

percentage\_outliers = 0 # or any other default value

print(f"Percentage of outliers in \033[1m{i}\033[0m column is \033[1m{round(percentage\_outliers, 2)}\033[0m%.")

l = ['CO(GT)', 'PT08.S1(CO)', 'C6H6(GT)', 'PT08.S2(NMHC)', 'NOx(GT)', 'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)', 'PT08.S5(O3)',

'T', 'RH', 'AH']

plt.figure(figsize = (20,30))

j = 1

for i in l:

plt.subplot(6, 2, j)

sns.boxplot(df[i])

plt.title("Boxplot of {}".format(i))

plt.xticks(rotation = 30)

j = j+1

plt.tight\_layout()

df.info()

# Select only numeric columns

numeric\_cols = df.select\_dtypes(include=[np.number])

# Calculate quartiles for numeric columns

Q1 = numeric\_cols.quantile(0.25)

Q3 = numeric\_cols.quantile(0.75)

IQR = Q3 - Q1

scale = 1.4

lower\_lim = Q1 - scale \* IQR

upper\_lim = Q3 + scale \* IQR

# Mask to remove rows with outliers

condition = ~((numeric\_cols < lower\_lim) | (numeric\_cols > upper\_lim)).any(axis=1)

# Apply mask to original DataFrame

df\_filtered = df[condition]

! pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip

df.reset\_index(drop = True, inplace = True)

report = ProfileReport(df)

#report

df\_filtered.reset\_index(drop = True, inplace = True)

report = ProfileReport(df\_filtered)

#report

df\_filtered.drop(['CO(GT)'] , axis = 1, inplace = True, errors = 'ignore')

df\_filtered.drop(['NOx(GT)'] , axis = 1, inplace = True, errors = 'ignore')

df\_filtered.drop(['C6H6(GT)'], axis = 1, inplace = True, errors = 'ignore')

df\_filtered.drop(['NO2(GT)'] , axis = 1, inplace = True, errors = 'ignore')

df\_filtered.head()

df\_filtered.reset\_index(drop = True, inplace = True)

report = ProfileReport(df\_filtered)

#report

month\_df\_list = []

day\_df\_list = []

hour\_df\_list = []

months = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November',

'December']

days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']

for month in months:

temp\_df = df\_filtered.loc[(df\_filtered['Month'] == month)]

month\_df\_list.append(temp\_df)

for day in days:

temp\_df = df\_filtered.loc[(df\_filtered['Weekday'] == day)]

day\_df\_list.append(temp\_df)

for hour in range(24):

temp\_df = df\_filtered.loc[(df\_filtered['Hour'] == hour)]

hour\_df\_list.append(temp\_df)

def df\_time\_plotter(df\_list, time\_unit, y\_col):

months = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November',

'December']

days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']

if time\_unit == 'M':

nRows = 3

nCols = 4

n\_iter = len(months)

elif time\_unit == 'D':

nRows = 2

nCols = 4

n\_iter = len(days)

elif time\_unit == 'H':

nRows = 4

nCols = 6

n\_iter = 24

else:

print('time\_unit must be a string equal to M,D, or H')

return 0

fig, axs = plt.subplots(nrows = nRows, ncols = nCols, figsize = (40, 30))

axs = axs.ravel()

for i in range(n\_iter):

data = df\_list[i]

ax = axs[i]

data.plot(kind ='scatter', x = 'DateTime', y = y\_col , ax = ax, fontsize = 24)

ax.set\_ylabel('Pollutant Concentration', fontsize = 30)

ax.set\_xlabel('')

if time\_unit == 'M':

ax.set\_title(y\_col + ' ' + months[i], size = 40)

elif time\_unit == 'D':

ax.set\_title(y\_col + ' ' + days[i], size = 40)

else:

ax.set\_title(y\_col + ' ' + str(i), size = 40)

ax.tick\_params(labelrotation = 60)

# set the spacing between subplots

plt.subplots\_adjust(left = 0.1, bottom = 0.1, right = 0.9, top = 0.9, wspace = 0.4, hspace = 0.5)

plt.show()

df\_time\_plotter(month\_df\_list, 'M', 'PT08.S3(NOx)')

df\_time\_plotter(day\_df\_list, 'D', 'PT08.S3(NOx)')

df\_time\_plotter(hour\_df\_list, 'H', 'PT08.S3(NOx)')

plt.figure(figsize = (18, 6))

sns.barplot(x = 'Month', y = 'PT08.S3(NOx)', data = df\_filtered)

plt.title('NOx Values Per Month')

plt.xticks(rotation = 90)

plt.show()

plt.figure(figsize = (18, 6))

sns.barplot(x = 'Weekday', y = 'PT08.S3(NOx)', data = df\_filtered)

plt.title('NOx Values Per Day of the Week')

plt.xticks(rotation = 90)

plt.show()

plt.figure(figsize = (18, 6))

sns.barplot(x = 'Hour', y = 'PT08.S3(NOx)', data = df\_filtered)

plt.title('NOx Values Per Hour')

plt.xticks(rotation = 90)

plt.show()

df\_final = df\_filtered.iloc[:, 5:]

df\_final

df.final.info()

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

from sklearn.preprocessing import StandardScaler

# Feature Selection (if applicable)

# Drop the 'PT08.S3(NOx)' column from the features

X = df\_final.drop(columns=['PT08.S3(NOx)'])

# Target Variable

y = df\_final['PT08.S3(NOx)']

# Train-Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Feature Scaling

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

print(X.shape)

print(y.shape)

def model\_assess(X\_train, X\_test, y\_train, y\_test, model, title = "Default"):

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

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

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

train\_mse = mean\_squared\_error(y\_train, y\_train\_pred)

train\_r2 = r2\_score(y\_train, y\_train\_pred)

test\_mse = mean\_squared\_error(y\_test, y\_test\_pred)

test\_r2 = r2\_score(y\_test, y\_test\_pred)

results = pd.DataFrame([title,train\_mse, train\_r2, test\_mse, test\_r2]).transpose()

results.columns = ['Method','Training MSE','Training R2','Test MSE','Test R2']

return y\_train\_pred, y\_test\_pred, results

def multi\_model\_assess(df, models, y\_predict):

all\_model\_results = [] #This will contain all model results for each dependent variable

all\_X\_test = []

all\_X\_train = []

all\_y\_test\_p = []

all\_y\_train\_p = []

all\_y\_train = []

#First loop will define dependent/independent variables and split data into test/training sets

n\_vars = len(y\_predict)

pbar = tqdm(range(n\_vars), desc = "Variable Processed", position = 0, leave = True) #Add progress bar

for dependent in y\_predict:

model\_results = [] #Array with dataframes for a given dependent variable

#Designate independent and dependent variables

x = df.drop([dependent], axis = 1)

y = df[dependent]

#Split data into test and training sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.2, random\_state = 42)

#Populate the array of observed values for the dependent variable

all\_y\_train.append(y\_train)

#Process each of the desired models

for model, model\_name in models:

y\_train\_pred,y\_test\_pred, results = model\_assess(X\_train, X\_test, y\_train, y\_test, model, title = model\_name)

model\_results.append(results)

all\_X\_test.append(X\_test)

all\_X\_train.append(X\_train)

all\_y\_test\_p.append(y\_test\_pred)

all\_y\_train\_p.append(y\_train\_pred)

all\_model\_results.append(model\_results)

pbar.update(1)

pbar.refresh()

pbar.close()

return all\_model\_results, all\_X\_test, all\_X\_train, all\_y\_test\_p, all\_y\_train\_p, all\_y\_train

#Initiate Different Regressors for ML model

lr = LinearRegression()

rf = RandomForestRegressor(n\_estimators = 100, max\_depth = 3, random\_state = 42)

gb = GradientBoostingRegressor(n\_estimators = 100, max\_depth = 3, random\_state = 42)

kn = KNeighborsRegressor()

ab = AdaBoostRegressor()

sv = SVR()

nn = MLPRegressor(hidden\_layer\_sizes = 500, solver = 'adam', learning\_rate\_init = 1e-2, max\_iter = 500)

models = [(lr,'Linear Regression'),

(rf,'Random Forest'),

(gb,'Gradient Boosting'),

(kn,'K-Neighbors'),

(ab,'Ada Boost'),

(sv,'SVR'),

(nn,'MLP')]

y\_predict = ['PT08.S1(CO)','PT08.S2(NMHC)','PT08.S3(NOx)','PT08.S4(NO2)','PT08.S5(O3)']

all\_model\_results, \_, \_, all\_y\_test\_p, all\_y\_train\_p, all\_y\_train = multi\_model\_assess(df\_final, models, y\_predict)

score\_df\_results = pd.concat(all\_model\_results[0], ignore\_index = True).sort\_values('Test R2', axis = 0, ascending = False)

score\_df\_results

score\_results\_test = pd.concat(all\_model\_results[0], ignore\_index = True)

score\_results\_test['Test R2'][0]

#Define column with model to predict

y\_predict = ['PT08.S1(CO)', 'PT08.S2(NMHC)', 'PT08.S3(NOx)', 'PT08.S4(NO2)', 'PT08.S5(O3)']

#Model names for plot titles

models = [

(lr,'Linear Regression'),

(rf,'Random Forest'),

(gb,'Gradient Boosting'),

(kn,'K Neighbors'),

(ab,'Ada Boost'),

(sv,'SVR'),

(nn,'MLP')]

#Make labels

def make\_labels(models):

names = []

for i in range(len(models)):

if len(models[i][1].split()) < 2:

names.append(models[i][1])

else:

names.append(''.join([s[0] for s in models[i][1].split()]))

return names

labelList = make\_labels(models)

#Specify color map to color different plots

cmap = plt.cm.get\_cmap('plasma')

slicedCM = cmap(np.linspace(0, 1, len(models)))

#Visualize results of linear regression

plt.rcParams.update({'font.size': 20})

nRows = 4

nCols = 2

def plot\_ML\_model(whichVar):

fig, axs = plt.subplots(nrows = nRows, ncols = nCols, figsize = (15, 30))

axs = axs.ravel()

df = pd.concat(all\_model\_results[whichVar], ignore\_index = True)

for k in range(7):

color = slicedCM[k]

yPred = all\_y\_train\_p[k + whichVar\*len(models)]

yMeas = all\_y\_train[whichVar]

label = labelList[k]

ax = axs[k]

#Make scatter plot of train set and regressor model

ax.scatter(x = yMeas, y = yPred, color = color, alpha = 0.5)

#Fit a first order polynomial (i.e. a straight line) to the regressor model

z = np.polyfit(yMeas, yPred, 1)

p = np.poly1d(z)

#Add labels and colors and stuff

val = df['Test R2'][k] #Get the r2 value from the model results dataframe

val = "{:.2f}".format(val)

ax.plot(yMeas, p(yMeas), "#b20cd7", label = label +"\nr\u00b2".format(2) + " = " + str(val))

ax.title.set\_text(models[k][1])

ax.set(xlabel = 'Train Concentration', ylabel = 'Predicted Concentration')

ax.label\_outer()

ax.legend(loc = "upper left")

ax.grid(color = 'black', linestyle = '--', linewidth = 0.5)

plt.show()

plot\_ML\_model(0)

score\_df\_results = pd.concat(all\_model\_results[1], ignore\_index = True).sort\_values('Test R2', axis = 0, ascending = False)

score\_df\_results

plot\_ML\_model(1)

score\_df\_results = pd.concat(all\_model\_results[2], ignore\_index = True).sort\_values('Test R2', axis = 0, ascending = False)

score\_df\_results

plot\_ML\_model(2)

score\_df\_results = pd.concat(all\_model\_results[3], ignore\_index = True).sort\_values('Test R2', axis = 0, ascending = False)

score\_df\_results

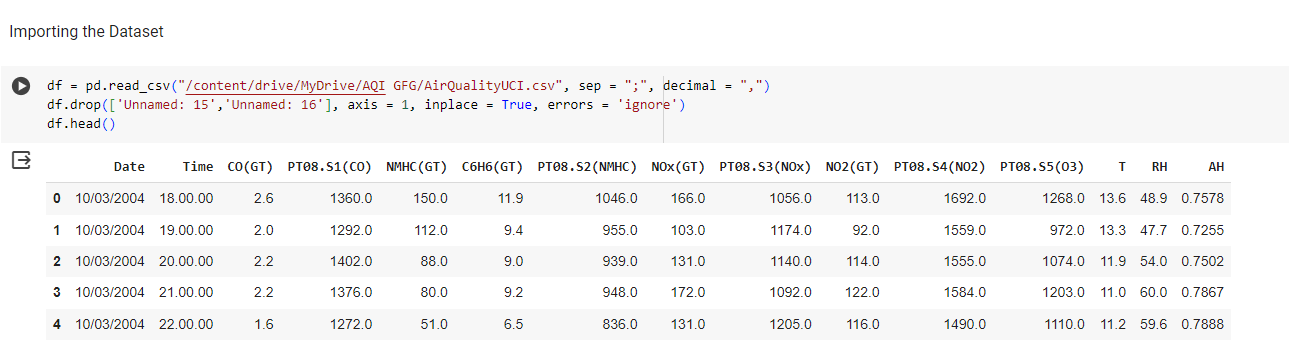
plot\_ML\_model(3)

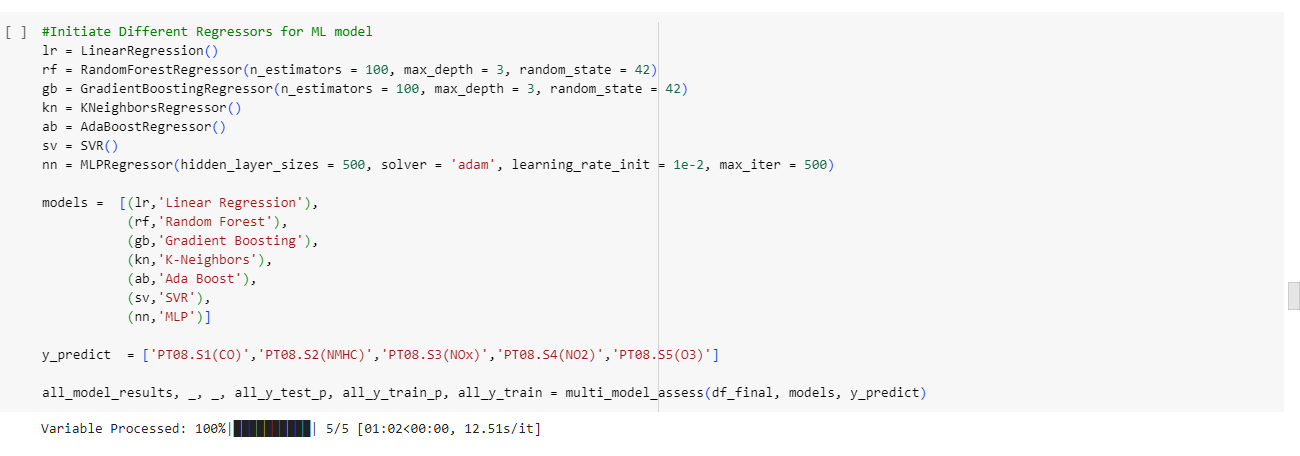
score\_df\_results = pd.concat(all\_model\_results[4], ignore\_index = True).sort\_values('Test R2', axis = 0, ascending = False)

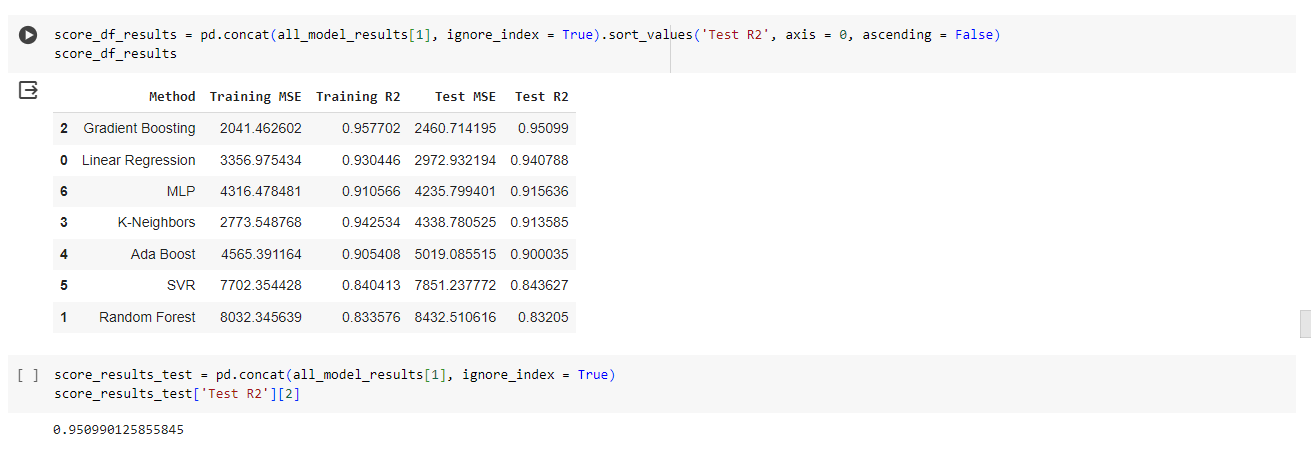
score\_df\_results

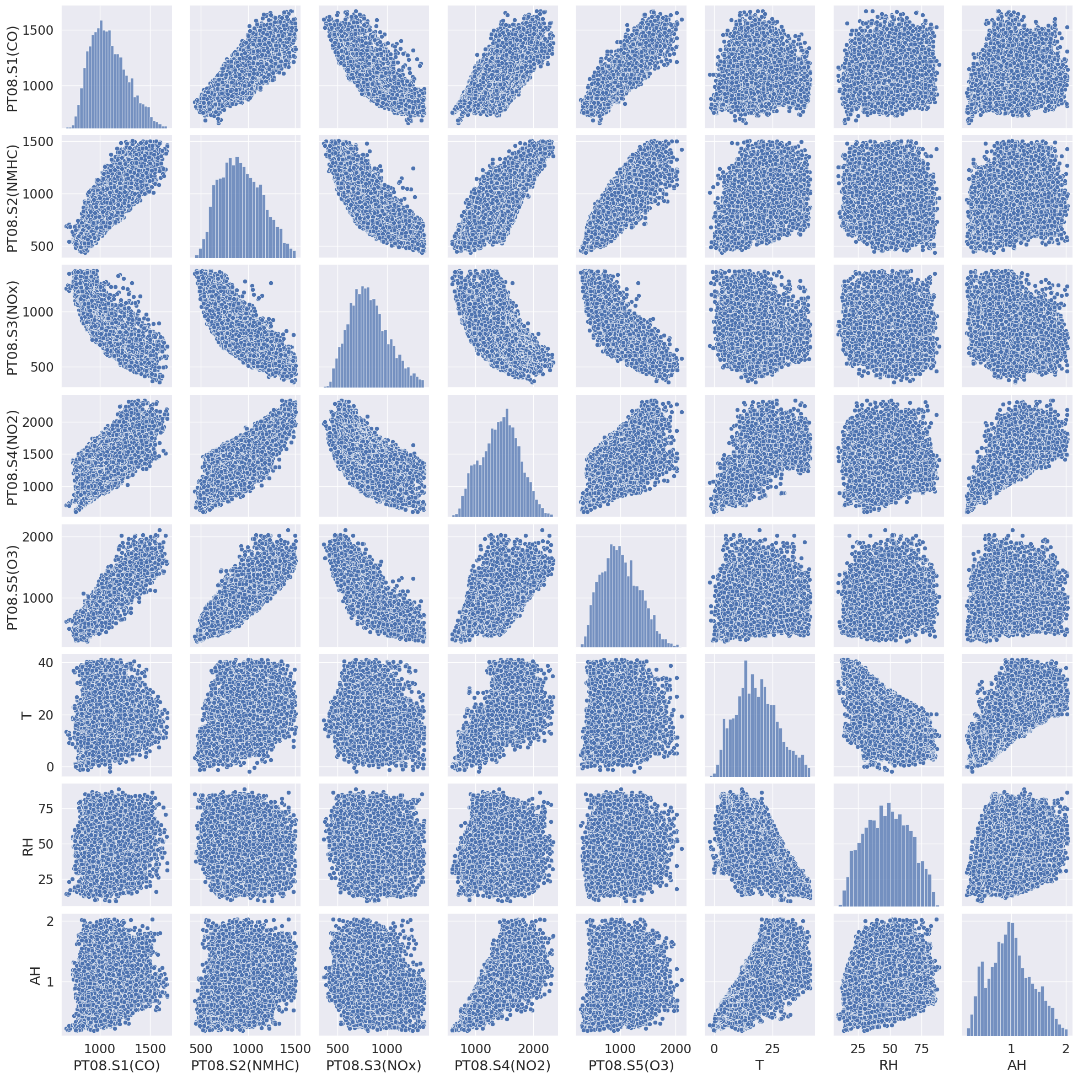
plot\_ML\_model(4)

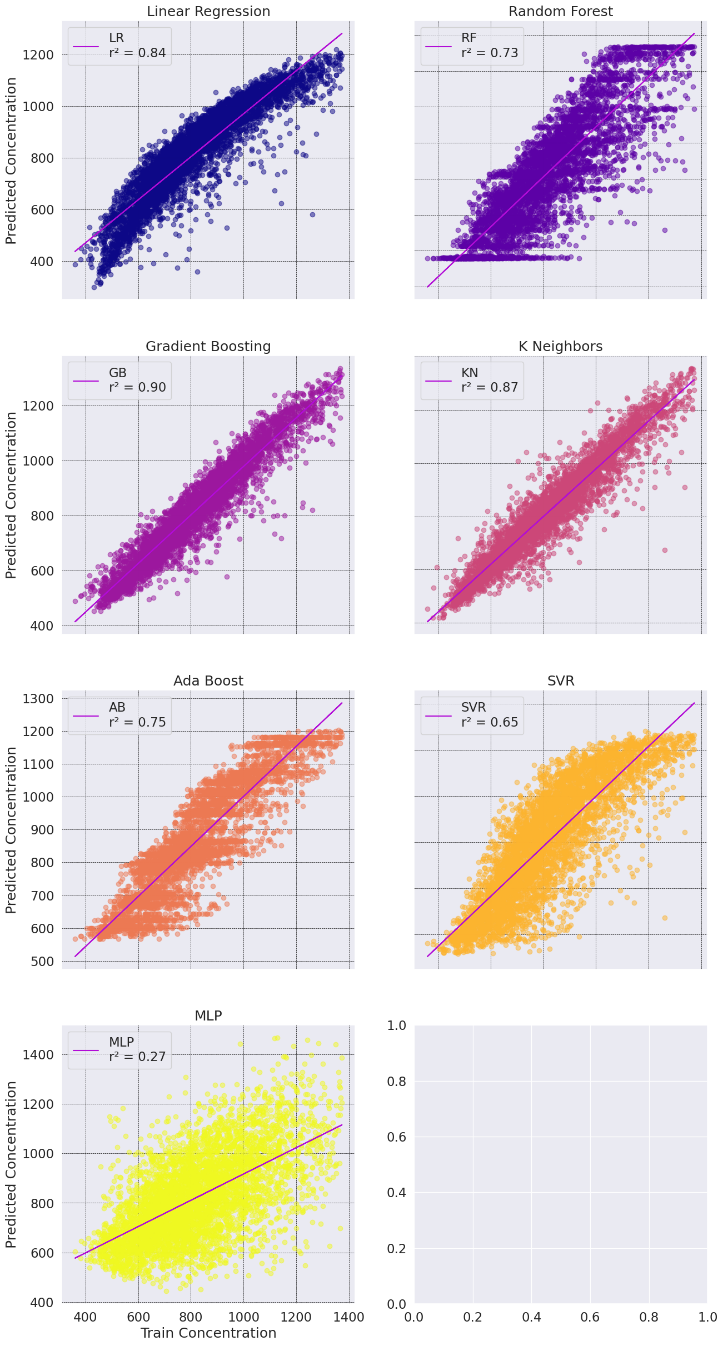
**9.2** **Screenshots**











**Chapter X**

**Bibliography**

**CHAPTER X**

**Bibliography**

## References

[1] "Machine Learning Approaches for Air Quality Index Prediction: A Comprehensive Review". Authors: Zhang, L., Liu, Q., Zhang, Z., & Zhang, L.

[2] "A Review of Machine Learning Techniques for Air Quality Prediction". Authors: Li, C., Ma, L., Hu, X., & Yao, Z.

[3] "Machine Learning-Based Air Quality Prediction: A Systematic Literature Review". Authors: Wang, Y., Huang, J., Chen, Y., & Yang, L.\*

[4] "Deep Learning Approaches for Air Quality Index Prediction: A Review". Authors: Xu, Y., Li, X., & Wang, L.

[5] "Advances in Machine Learning-Based Air Quality Forecasting: A Review". Authors: Wu, Q., Wang, Z., & Sun, H.

[6] "Machine Learning-Based Air Quality Prediction Models: A Review and Comparison". Authors: Chen, X., Sun, C., Li, S., & Zhang, H.

[7] "Machine Learning Approaches for Air Quality Index Prediction: Current Trends and Future Directions". Authors: Liu, Y., Wang, J., Zhang, W., & Zhao, Y.

[8] "A Review of Ensemble Learning Techniques for Air Quality Index Prediction". Authors: Huang, Y., Li, Z., Zhou, L., & Wang, Q.

[9] "Recent Advances in Machine Learning-Based Air Quality Forecasting: A Comprehensive Review". Authors: Zhang, J., Liu, Y., Wang, H., & Chen, Z.

[10] "Machine Learning Approaches for Urban Air Quality Monitoring: A Systematic Review". Authors: Yang, H., Li, J., Wang, Y., & Zhang, X.

[11] "Air Pollution Modeling and Its Application XXIV" edited by Douw G. Steyn and Nadine Chaumerliac: This book covers various aspects of air pollution modeling, including techniques for predicting AQI.

[12] "Air Quality Modeling: Theories, Methodologies, Computational Techniques, and Available Databases and Software" by Paolo Zannetti.

[13] "Air Pollution: Measurement, Modelling and Mitigation" by Jeremy Colls.

[14] "Atmospheric Chemistry and Physics: From Air Pollution to Climate Change" by John H. Seinfeld and Spyros N. Pandis: While this book primarily focuses on atmospheric chemistry and physics, it includes discussions on air pollution modeling and its connection to climate change, which can be beneficial for understanding AQI prediction.

[15] "Air Pollution Control Engineering" by Noel de Nevers: Although more focused on control engineering aspects, this book provides fundamental knowledge about air pollution, which is essential for developing models and predicting AQI.

[16] "Air Pollution Modeling and Its Application XXIII" edited by Douw G. Steyn and Nadine Chaumerliac

[17] "Air Quality Management" by Ian Colbeck

[18] "Air Pollution Control Technology Handbook" by Karl B. Schnelle Jr. and Charles A. Brown

[19] "Air Quality Assessment and Management: A Practical Guide" by Vladimir Kucera

[20] "Air Pollution: Measurement, Modelling and Mitigation" edited by Renato Baciocchi, Francesca Miani, and Matteo Vinci

[21] "Air Quality: Monitoring, Measuring, and Modeling Environmental Hazards" by S. K. Goyal and V. K. Jain

[22] "Atmospheric Chemistry and Physics: From Air Pollution to Climate Change" by John H. Seinfeld and Spyros N. Pandis

[23] "Introduction to Environmental Engineering and Science" by Gilbert M. Masters and Wendell P. Ela

[24] "Urban Air Quality: Measurement, Modelling and Management" edited by Ranjeet S. Sokhi and Alexander Baklanov

[25] "Fundamentals of Air Pollution" by Daniel A. Vallero

## Website References

* <https://docs.python.org/3/tutorial/>
* <https://www.w3schools.com/python/>
* <https://www.programiz.com/python-programming>
* [https://www.learnpython.org/p](https://www.learnpython.org/)ython