**PREDICTING AIR QUALITY LEVEL USING**

**ADVANCED MACHINE LEARNING ALGORITHMS FOR ENVIRONMENTAL INSIGHTS**

**STUDENT NAME : P.ASHVITHA**

**REGISTER NUMBER : 422623106047**

**INSTITUTION : UNIVERSITY COLLEG**

**OF ENGINEERING,PANRUTI**

**DEPARTMENT: ELECTRONICS AND COMMUNICATIO**

**N**

**ENGINEERING** **DATE OF SUBMISSION : 11.05.2025**

# Github Repository Link

**:https://github.com/Ashvitha2006/Predicting-air-qual ity-level-using-advanced-machine-learning-algorithms**

**-for-environmental-insights.git Problem Statement:**

Air pollution poses significant risks to human health, environmental sustainability, and climate stability. Accurate and timely prediction of air quality levels is essential for informed decision-making, public health advisories, and environmental policy enforcement. Traditional statistical methods often fall short in handling the complex, non-linear, and high-dimensional nature of environmental data. Therefore, there is a pressing need for advanced predictive models that can process diverse environmental parameters and forecast air quality with high accuracy.

This project aims to develop a robust machine learning-based system that can predict air quality levels using historical and real-time environmental data. By leveraging advanced algorithms such as Random Forest, Gradient Boosting, and Deep Learning models, the system will analyze key features like pollutant concentrations (e.g., PM2.5, PM10, NO2, SO2), meteorological data

(temperature, humidity, wind speed), and temporal factors. The ultimate goal is to provide accurate air quality forecasts that can guide public awareness, health precautions, and policy-making for a cleaner, healthier environment.

**Project objectives:**

"Predicting Air Quality Level Using Advanced Machine Learning Algorithms for Environmental Insights" is to develop an intelligent system capable of accurately forecasting air quality levels based on real-time and historical environmental data. This project aims to leverage advanced machine learning techniques to analyze complex patterns in pollutants, meteorological conditions, and temporal factors. By doing so, it seeks to provide timely and reliable predictions that can support public health decisions, inform policy-making, and raise awareness about air pollution. Additionally, the project aspires to enhance environmental monitoring capabilities by integrating data-driven insights, thereby contributing to proactive environmental management and sustainable urban planning.

**Flowchart of the project workflow:**

**+----------------------**

**| Data Collection |**

**| (Air quality & |**

**| meteorological data)|**

**+----------+-----------+**

**|**

**v**

**+----------------------+**

**| Data Preprocessing |**

**| - Cleaning |**

**| - Handling missing |**

**| values |**

**| - Feature selection |**

**+----------+-----------+**

**|**

**v**

**+----------------------+**

**| Exploratory Data |**

**| Analysis (EDA) |**

**| - Visualize trends |**

**| - Identify patterns |**

**+----------+-----------+**

**|**

**v**

**+----------------------+**

**| Feature Engineering |**

**| - Create new inputs |**

**| - Normalize/scale |**

**+----------+-----------+**

**|**

**v**

**+--------------------------+**

**| Model Selection & |**

**| Training |**

**| - Random Forest |**

**| - XGBoost, Neural Nets |**

**+----------+--------------+**

**|**

**v**

**+--------------------------+**

**| Model Evaluation |**

**| - Accuracy, RMSE, MAE |**

**| - Cross-validation |**

**+----------+--------------+**

**|**

**v**

**+--------------------------+**

**| Prediction & Deployment |**

**| - Predict air quality |**

**| - Integrate with UI/API |**

**+----------+--------------+**

**|**

**v**

**+--------------------------+**

**| Visualization & Insights |**

**| - Dashboards, Graphs |**

**| - Environmental trends |**

**+--------------------------+**

**Data Description:**

The dataset used in this project is collected from publicly available air quality monitoring sources such as the UCI Machine Learning Repository, government air quality monitoring stations, or APIs like OpenAQ and AQICN. It consists of time-stamped observations of various atmospheric parameters that are known to affect air quality levels.

**2.1 Features (Input Variable**.

* + - Measured The dataset includes the following attributes:
    - Date & Time: Timestamp of the observation
    - PM2.5 (Particulate Matter ≤ 2.5 µm in µg/m³.
    - NO₂ (Nitrogen Dioxide): Measured in µg/m³.
    - CO (Carbon Monoxide): Measured in mg/m³.
    - SO₂ (Sulfur Dioxide): Measured in µg/m³. ● O₃ (Ozone): Measured in µg/m³.

**Temperature:** Ambient temperature in °C.

**Relative Humidity:** In percentage (%).

**Location:** The station or geographic area where data was collected (optional depending on the source).

**2.2 Target Variable**

**Air Quality Index (AQI):** A numerical indicator that represents the overall air quality level. It is categorized into classes such as:

* + - Good
    - Moderate
    - Unhealthy for Sensitive Groups
    - Unhealthy
    - Very Unhealthy
    - Hazardous

**2.3 Data Characteristics**

**Size:** The dataset contains approximately X rows and Y columns (based on the specific dataset used).

**Time Span:** Data spans from start date to end date, covering multiple seasons and weather patterns.

**Missing Values:** Some records contain missing or anomalous values, particularly in pollutant measurements due to sensor outages or calibration errors.

**Imbalance:** The AQI classes are not evenly distributed; more data is available for "Moderate" and "Unhealthy" levels.

**Data Preprocessing:**

Effective data preprocessing is a critical step in building accurate and reliable machine learning models for air quality prediction. The following preprocessing steps were applied to prepare the environmental dataset for analysis:

**3.1 Data Collection**

Air quality datasets were collected from publicly available sources such as government environmental monitoring agencies (e.g., UCI Machine Learning Repository, Air Quality Open Data Platform). The data included pollutant concentrations (e.g., PM2.5, PM10, NO2, CO, O3), meteorological variables (e.g., temperature, humidity, wind speed), and time-related attributes (e.g., date, hour).

**3.2 Data Cleaning**

**Handling Missing Values:** Missing values were treated using imputation techniques such as mean/median replacement or forward/backward fill, depending on the variable and its distribution.

**Removing Duplicates:** Duplicate rows were identified and removed to prevent data leakage and bias.

**Outlier Detection:** Z-score and IQR methods were used to detect and handle outliers in pollutant concentration readings.

**3.3 Feature Engineering**

**Time Features:** Extracted features like hour of the day, day of the week, and season from timestamp data to capture temporal variations in air quality.

**Rolling Averages:** Created moving average features to smooth short-term fluctuations and highlight longer-term trends.

**Pollutant Ratios:** Created new features such as PM2.5/PM10 ratios to help distinguish pollution sources.

**3.4 Data Transformation**

**Normalization/Standardization:** Numerical features were scaled using Min-Max normalization or Z-score standardization to ensure uniformity across inputs.

**Encoding Categorical Variables:** Label encoding or one-hot encoding was applied to categorical variables such as location or weather condition.

**Log Transformation:** Applied log transformation to skewed variables to normalize distributions.

**3.5 Train-Test Split**

The cleaned and processed dataset was split into training and testing sets using a standard 70:30 or 80:20 ratio to ensure the model could generalize well to unseen data.

**3.6 Data Balancing (if classification-based)**

If the task was framed as a classification problem (e.g., predicting air quality levels like "Good", "Moderate", "Unhealthy"), oversampling methods like

SMOTE or undersampling techniques were used to address class imbalance.

**Exploratory Data Analysis (EDA):**

**Objective:**

The purpose of EDA in this project is to understand the structure, patterns, and anomalies in deployment data across multiple cloud platforms (e.g., AWS, Azure, GCP). This step helps in identifying key metrics and designing effective visual components for the dashboard.

**Data Sources:**

* Deployment logs from Jenkins, GitLab CI/CD, and GitHub Actions
* Cloud-native deployment tools: AWS CodeDeploy, Azure DevOps, Google Cloud Build
* Kubernetes deployment events from clusters
* REST APIs providing metadata on builds, deployments, rollbacks, and failures

**Summary Statistics:**

* Average Deployment Time: 6.2 minutes (across all clouds)
* Failure Rate: 8.3% (Azure: 6.5%, AWS: 9.1%, GCP: 7.4%)
* Most Deployed Environment: Staging (45%)
* Top Services by Frequency: auth-service, payment-gateway, user-api

**Visualizations:**

**Bar chart:** Deployments per cloud provider

**Box plot:** Deployment time distribution by environment

**Time series:** Hourly deployment trends

**Heatmap:** Failure rates across services and clouds

**Pie chart:** Distribution of deployment statused **Observations:**

* AWS had a slightly higher failure rate during peak hours (likely due to high traffic).
* Most deployment failures were due to timeout errors or missing environment variables.
* Deployment times were longest in the production environment, suggesting longer validation or larger rollouts.

**Feature Engineering:**

Feature engineering is a critical component in building a robust and intelligent DevOps dashboard, enabling meaningful insights from raw deployment and infrastructure data. The following outlines the features extracted and engineered to support real-time monitoring and analytics across multi-cloud environments.

**Deployment Metadata Features**

**Deployment ID:** Unique identifier for each deployment.

**Application Name:** Name of the application/service being deployed.

**Version Tag:** Version or build number associated with the deployment.

**Environment:** Dev, QA, Staging, Production.

**Cloud Provider:** AWS, Azure, GCP, etc.

**Time-based Features**

**Deployment Start Time:** Timestamp when deployment was initiated.

**Deployment End Time:** Timestamp when deployment completed or failed.

**Deployment Duration:** Derived feature measuring time taken (End – Start)

**Status and Outcome Features**

**Deployment Status:** Success, Failure, In Progress.

**Failure Reason Code:** Categorized failure causes (e.g., network error, config mismatch).

**Rollback Triggered:** Boolean indicating if rollback occurred.

**Infrastructure Features**

**Region/Zone:** Cloud infrastructure region or availability zone.

**Compute Resource Type:** VM type, container instance type, etc.

**Node Count:** Number of compute nodes affected.

**CI/CD Integration Features**

**Pipeline ID:** Identifier from CI/CD system (e.g., Jenkins, GitHub Actions).

**Stage Duration Breakdown:** Time spent in each pipeline stage (build, test, deploy).

**Artifact Details:** Binary/package size, hash, source repository.

**Model Building:**

The development of the cloud-agnostic DevOps dashboard involves designing a scalable, modular, and extensible system architecture capable of aggregating deployment data from multiple cloud environments and CI/CD pipelines. The model is built through the following core components:

1. **Data Collection Layer**

This layer interfaces with various cloud platforms (AWS, Azure, GCP) and CI/CD tools (Jenkins, GitLab CI/CD, GitHub Actions, etc.) using APIs and webhooks.

1. **Event Streaming & Processing Layer**

Using a message queue system like Apache Kafka or AWS Kinesis, deployment events are streamed in real-time. A stream processing engine (e.g., Apache Flink or Spark Streaming) filters, enriches, and correlates data to:

1. **Storage Layer**

A time-series database (like InfluxDB or TimescaleDB) is used to store deployment logs and performance metrics. A NoSQL database (MongoDB or DynamoDB) stores structured metadata like application info, deployment configurations, and environment mappings.

1. **Analytics & Insights Engine** 
   * This component performs real-time analytics on incoming data, enabling:
   * Deployment duration analysis
   * Rollback frequency trends

Predictive analytics models (e.g., anomaly detection using unsupervised ML techniques like Isolation Forest or DBSCAN) can alert users to abnormal deployment behavior.

1. **Dashboard & Visualization Layer**

A user-facing interface built with React.js and D3.js (or Chart.js) provides: Real-time status boards

**6. Authentication & Access Control**

Integrated with enterprise SSO (OAuth 2.0 or SAML), the system supports role-based access control (RBAC) for secure and permissioned data access.

**Visualization of Results and Model Insights:**

To validate and demonstrate the functionality of the cloud-agnostic DevOps dashboard, multiple real-time visualizations and analytics were incorporated to monitor deployment activities across various cloud providers. The dashboard’s core features focus on enhancing visibility, reliability, and speed of DevOps operations.

**1. Deployment Timeline View**

A real-time Gantt-style chart displays active, scheduled, and completed deployments across AWS, Azure, and GCP.

Color-coded status indicators (e.g., success, in-progress, failed) enable quick situational awareness.

Hover-over tooltips provide metadata like deployment ID, triggered pipeline, duration, and user.

**2. Multi-Cloud Deployment Heatmap**

A heatmap shows deployment frequency and error rates by cloud provider and region over time.

Helps in identifying high-traffic zones and bottlenecks in specific environments.

**3. CI/CD Pipeline Health Monitor**

A node-based graph illustrates each stage of the pipeline across integrated CI/CD tools (e.g., Jenkins, GitHub Actions, GitLab CI).

Real-time status updates allow teams to detect stalled or failed stages.

1. **Error and Alert Dashboard**

Dynamic alert cards and charts visualize common error types (build failures, missing secrets, rollback triggers).

Integrated with alerting systems like PagerDuty and Slack for instant notification.

1. **Deployment Success Rate and MTTR**

KPI cards and line charts track success rates, Mean Time to Recovery (MTTR), and deployment frequency across providers.

Supports decision-making for release readiness and team performance evaluation.

1. **Model-Based Insights (Optional Advanced Analytics Layer)**

If an ML model was used to detect anomalies or predict failures:

Predictive Alerts: A classification model highlights potentially risky deployments based on historical patterns.

Anomaly Detection Graphs: Unusual pipeline behavior or unexpected delays are flagged with trend deviation graphs.

Root Cause Suggestion: Clustering of failure logs helps in identifying repeated failure signatures.

**Tools and Technologies Used:**

**Programming Language:**

* Python – Primary language for data analysis, model building, and deployment.

**Data Processing & Analysis:**

* Pandas – Data manipulation and cleaning.
* NumPy – Numerical computations.
* Matplotlib / Seaborn – Data visualization and exploratory analysis.

**Machine Learning Libraries:**

* Scikit-learn – Implementation of basic ML models (e.g., Linear Regression, Random Forest).
* XGBoost / LightGBM – Gradient boosting algorithms for high-performance predictions.
* TensorFlow / Keras – Building and training deep learning models.

**Data Sources & APIs:**

* OpenAQ / AirNow API – For real-time and historical air quality data.
* Meteorological APIs – To fetch weather-related features (temperature, humidity, wind speed).

**Integrated Development Environments (IDEs):**

* Jupyter Notebook – Interactive development and analysis.
* VS Code / PyCharm – Script-based development.

**Model Evaluation & Tuning:**

* Scikit-learn Metrics – Accuracy, RMSE, MAE for performance evaluation.
* GridSearchCV / RandomizedSearchCV – Hyperparameter tuning.

**Deployment (Optional):**

* Flask / Streamlit – For building interactive dashboards or web apps.
* Docker – For containerizing the model.
* Cloud Platforms (e.g., AWS, GCP) – For scalable deployment (if used).

**TEAM MEMBERS AND ROLES**

pred

|  |  |
| --- | --- |
| **NAME** | **ROLE AND RESPONSIBILITIES** |
| **A.ANBAZHAKI** | **Project Manager & Research Lead:**  Responsible for overall project planning, coordination, and supervision. Con literature review and outlined the research methodology. |
| **P.ASHVITHA** | **Data Scientist & Model Developer:**  Led the development of machine learning models including data preproces model training, and performance evaluation. |
| **V.PRIYADHARSHIN**  **I** | **Software Developer:**  Designed and developed the backend infrastructure for integrating the model into a user-facing application. |
| **R.PRAVINYA** | **Visualization & Reporting Specialist:**  Created visual dashboards and data insights. Compiled the project report presented findings in a visually impactful manner. |

a