Real-Time Air Quality Monitoring and Pollution Prediction using Big Data Analytics

A PROJECT REPORT

Submitted by

Kedar Mohit	**CB.AI.U4AID23045**
Chaitanya Mahesh	**CB.AI.U4AID23058**
Mahesh Reddy	**CB.AI.U4AID23059**
Kowshik Reddy	**CB.AI.U4AID23060**

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

21AID302 – BIG DATA ANALYTICS

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

COIMBATORE – 112

OCTOBER 2025

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

COIMBATORE – 112

BONAFIDE CERTIFICATE

This is to certify that the report entitled “Real-Time Air Quality Monitoring and Pollution Prediction using Big Data Analytics” submitted by Chaitanya Mahesh (CB.AI.U4AID23058), Mahesh Reddy (CB.AI.U4AID23059), Kowshik Reddy(CB.AI.U4AID23060), Kedar Mohit (CB.AI.U4AID23045) for the award of the Degree of Bachelor of Technology in the “**ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**” is a bonafide record of the work carried out under my guidance and supervision at Amrita School of Artificial Intelligence , Coimbatore.

Project Guide : Dr.SREEJA B P
Assistant Professor(Sr.Gr.)

Submitted for the University Examination held on ______________

Examiner 1 **Examiner 2**

TABLE OF CONTENT

1. ABSTRACT

Air pollution remains one of the most critical environmental challenges affecting human health, ecosystems, and climate balance, and with rapid urbanization and industrial expansion, monitoring and predicting the Air Quality Index (AQI) has become vital for policy formulation and public health management; therefore, this project proposes a Big Data–driven analytical framework for real-time air quality monitoring and pollution prediction using Apache Spark and the Hadoop Distributed File System (HDFS), where the system integrates large-scale data ingestion, preprocessing, and machine learning model training through Spark's distributed computing capabilities to ensure high performance and scalability; by leveraging both historical and real-time data from OpenAQ, we conduct pollutant forecasting for key indicators such as PM10, PM2.5, O_3, NO_2, CO, and SO_2 using a Random Forest Regression model implemented in Spark MLlib, and the experimental results demonstrate that the framework achieves scalable performance, efficient and parallelized data processing, and strong prediction accuracy with R^2 values greater than 0.8 for all major pollutants, ultimately highlighting how Big Data Analytics can power intelligent, data-driven environmental decision-making systems that support proactive pollution control strategies and protect public health.

2. INTRODUCTION

Air pollution has emerged as a serious global crisis, impacting both developed and developing nations. Exposure to high concentrations of pollutants such as particulate matter (PM10 and PM2.5), nitrogen dioxide (NO_2), carbon monoxide (CO), sulphur dioxide (SO_2), and ozone (O_3) has been linked to a wide range of health issues including respiratory illnesses, cardiovascular diseases, and neurological disorders. According to the World Health Organization (WHO), air pollution contributes to millions of premature deaths every year, making it one of the leading environmental health risks worldwide.

Real-time monitoring of the Air Quality Index (AQI) is therefore essential for providing timely and actionable insights to governments, environmental agencies, and the public. It enables preventive measures to be taken before pollution levels reach hazardous thresholds. However, the continuous data streams generated by hundreds of monitoring stations worldwide result in extremely large and complex datasets, posing major challenges in terms of storage, computation, and real-time analysis.

Traditional data management systems are not designed to efficiently handle the volume, velocity, and variety of air quality data. This is where Big Data technologies come into play. Apache Spark offers distributed data processing capabilities that can efficiently manage and analyze petabyte-scale datasets, while Hadoop's Distributed File System (HDFS) provides scalable, fault-tolerant storage infrastructure. The combination of these tools facilitates the creation of an automated, end-to-end data processing pipeline that supports real-time data ingestion, cleaning, feature extraction, and pollutant prediction.

By leveraging Big Data Analytics and Machine Learning together, this project transforms raw environmental data into valuable predictive insights. Such integration not only enhances the understanding of pollution patterns but also supports smarter urban planning, early warning systems, and data-driven policymaking aimed at improving air quality and safeguarding public health.

3. LITERATURE SURVEY

Several studies have been conducted on air quality monitoring and prediction using data-driven and machine learning approaches. Kang et al. (2018), in their paper "Air Quality Prediction: Big Data and Machine Learning Approaches", reviewed various data-driven models used for Air Quality Index (AQI) prediction and highlighted the growing importance of Big Data platforms in achieving real-time, scalable environmental monitoring. Their work emphasized how traditional statistical models are being replaced by data-centric techniques that can process and analyze large volumes of sensor data efficiently.

Bosco and Kowsalya (2024), in their study titled "A Novel Approach for Air Pollution Prediction Using Machine Learning Techniques," implemented algorithms such as Logistic Regression and K-Nearest Neighbors (KNN) for AQI prediction. They demonstrated that careful preprocessing and feature scaling significantly improved the predictive accuracy of even relatively simple models. Their findings showed that effective data preparation plays a crucial role in environmental prediction systems.

Similarly, Kothandaraman et al. (2022), in their work "Intelligent Forecasting of Air Quality and Pollution Prediction Using Machine Learning," compared multiple machine learning models including Random Forest, XGBoost, and AdaBoost for PM2.5 prediction. Their study concluded that ensemble learning techniques provide higher accuracy and lower error rates than individual models, making them highly suitable for real-world pollution forecasting.

In summary, while existing studies have successfully demonstrated the application of machine learning to air quality prediction, most of them focused on localized datasets or smaller regions. Only a few explored the integration of Big Data frameworks to process large-scale, heterogeneous air quality data from multiple monitoring stations. This gap in handling global, high-volume environmental data has motivated our research to develop a scalable Spark-based framework that leverages distributed computation and machine learning for efficient and accurate air quality forecasting.

4. MOTIVATION / GAP ANALYSIS

Although several advanced systems have been proposed for monitoring air quality, many of them face significant challenges in scalability, data integration, and consistency. Existing environmental datasets are often fragmented, unstructured, and region-specific, which makes large-scale analysis difficult. Traditional analytics systems are not designed to efficiently process such massive datasets, especially when they involve multiple pollutants and continuous time-series readings from diverse monitoring stations. Additionally, most available APIs, including Open-AQ, are limited in terms of reliability and response rate, often leading to incomplete or delayed data retrieval. In our case, attempts to integrate real-time ingestion from the OpenAQ API were hindered by inconsistent responses, slow performance, and frequent access restrictions.

Because of these constraints, this project focuses primarily on historical data analysis rather than live data streaming. By utilizing Big Data technologies such as Apache Spark and Hadoop HDFS, the system aims to handle large volumes of historical AQI data efficiently, performing distributed preprocessing, cleaning, and pollutant prediction. This approach ensures scalability, robustness, and accurate forecasting without depending on unstable real-time API connections.

The gap identified is the lack of a stable, large-scale framework capable of processing and predicting air quality trends from global historical datasets using Big Data techniques. While many studies emphasize real-time systems, few focus on scalable and efficient offline batch processing of multi-year air quality data.

The motivation behind this project is to bridge this gap by developing a **u**nified Big Data analytical framework that can:

- Efficiently process large-scale historical AQI data collected from multiple monitoring stations.
- Utilize distributed machine learning models for pollutant-level forecasting.
- Enable flexible scaling and future integration with real-time ingestion once API reliability improves.

5. METHODOLOGY

The system workflow for the proposed air quality monitoring and prediction framework consists of five major stages — data collection, data preprocessing, feature engineering, model training, and evaluation and storage**.** Each stage is carefully designed to ensure scalability, efficiency, and accuracy while handling large volumes of air quality data.

1. Data Collection:
   The data used in this project is sourced from OpenAQ, a global open-access platform that provides both historical and real-time air quality data. However, due to limitations in the API's response time and reliability, the system primarily focuses on historical data ingestion. The dataset contains pollutant readings such as PM2.5, PM10, NO_2, SO_2, CO, and O_3, along with spatial and temporal attributes collected from multiple monitoring stations. These datasets are stored in the Hadoop Distributed File System (HDFS) to enable distributed access and processing.
2. Data Preprocessing:
   The preprocessing phase is implemented using Apache Spark, which allows

parallel and distributed operations on large datasets. The following steps are performed to prepare the data for analysis:

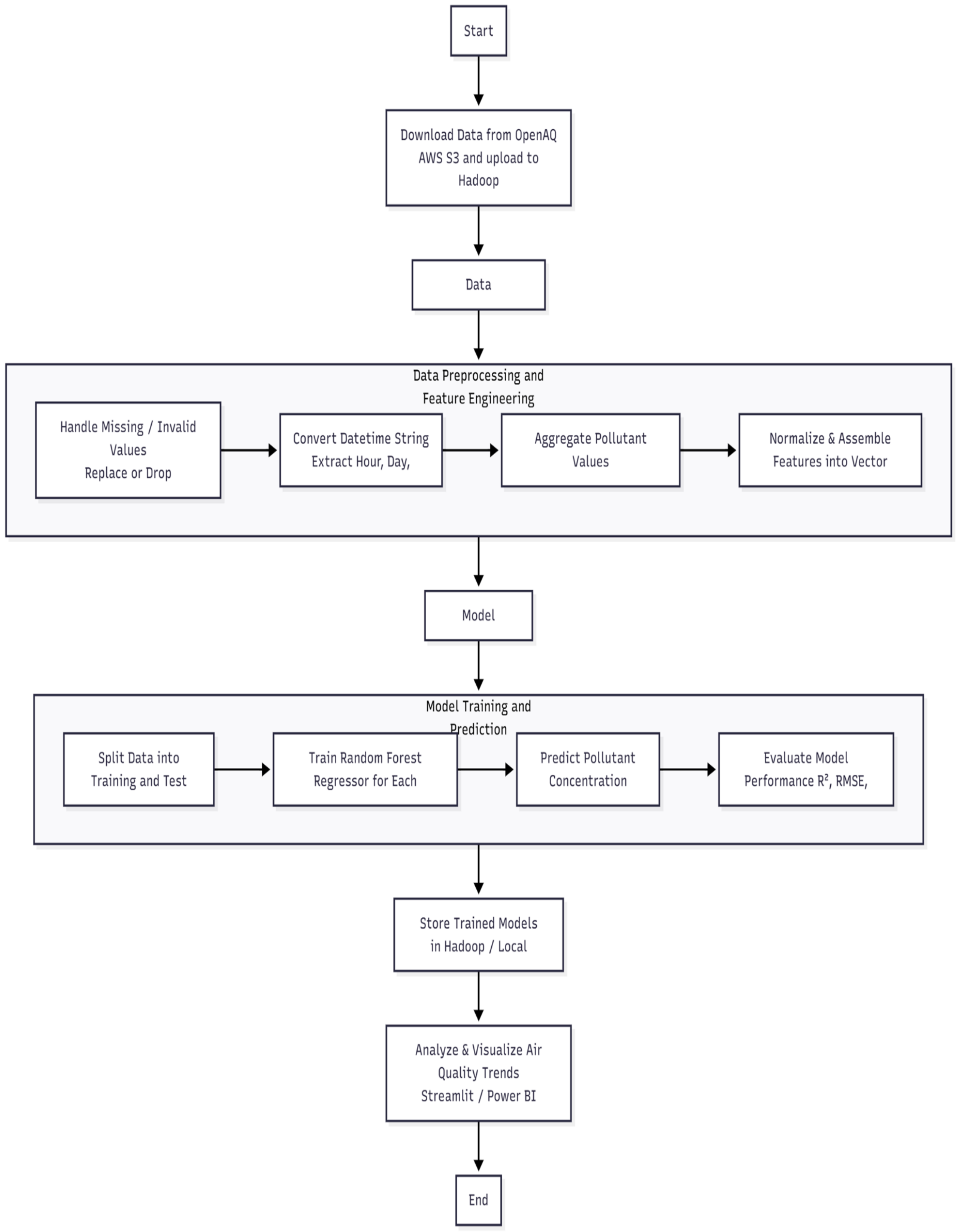
- Handling missing values: Null entries in pollutant readings are filled using a forward-fill approach with Spark's last() window function.
- Timestamp conversion and sorting: The datetime column is converted into a proper timestamp format and sorted by both location ID and time to maintain chronological order.
- Feature extraction: Temporal features such as hour, day, month, and weekday are derived from the datetime field to capture periodic variations in pollution levels.
- Outlier clipping: Pollutant readings are capped between 0 and 1000 to eliminate extreme or erroneous sensor values.
- Normalization: Pollutant concentrations are scaled by dividing by 1000 to standardize the input range for model training.

3. Feature Engineering:
   In this stage, additional features are derived to improve the predictive capability of the model.

   - Spatial features: Each monitoring station is identified using a unique location_id, allowing the model to learn regional pollution characteristics.
   - Temporal features: Time-based attributes, particularly hour of the day, help identify daily pollution patterns caused by traffic and industrial activity.
   - Correlated features: Relationships between pollutants are implicitly captured through multivariate regression, helping the model understand how changes in one pollutant affect others.

4. Model Training:
   The predictive modeling is carried out using Random Forest Regressor from Spark MLlib. Each pollutant (PM2.5, PM10, O_3, CO, NO_2, and SO_2) is modeled separately, where the target variable is the next-hour concentration level predicted using the current readings and time-based features. The dataset is split into training (80%) and testing (20%) sets. Key evaluation metrics include the Coefficient of Determination (R^2) to assess model accuracy and the Mean Absolute Error (MAE) to measure prediction deviation. Random Forest is chosen for its robustness, nonlinearity handling, and scalability across distributed nodes.
5. Evaluation and Storage:
   Once the models are trained, they are evaluated on the test dataset, and performance metrics are computed. Each trained model, along with its

corresponding metrics, is saved back to HDFS for future retrieval and deployment.



Start
Download Data from OpenAQ AWS S3 and upload to Hadoop
Data
Data Preprocessing and Feature Engineering
Handle Missing / Invalid Values Replace or Drop
Convert Datetime String Extract Hour, Day,
Aggregate Pollutant Values
Normalize & Assemble Features into Vector
Model
Model Training and Prediction
Split Data into Training and Test
Train Random Forest Regressor for Each
Predict Pollutant Concentration
Evaluate Model Performance R², RMSE,
Store Trained Models in Hadoop / Local
Analyze & Visualize Air Quality Trends Streamlit / Power BI
End

Fig1: Work Flowchart

6. IMPLEMENTATION

The implementation of this project integrated several Big Data technologies, each serving a specific role in building a scalable air quality prediction system. OpenAQ was chosen as the primary data source for both real-time and historical air quality data. Since the OpenAQ API showed performance issues and limited reliability, the historical data was downloaded directly from AWS S3 using the AWS Command Line Interface (CLI), which allowed efficient and secure data transfer from the cloud.

The downloaded dataset was stored in the Hadoop Distributed File System (HDFS), which served as the main storage layer. HDFS provided distributed storage, fault tolerance, and fast data access for subsequent processing. A total of 28 GB of disk space was allocated for HDFS on the macOS system to manage data files, intermediate outputs, and trained models efficiently.

Apache Spark acted as the core processing engine, performing all data preprocessing, cleaning, and transformation tasks using Spark DataFrames and Spark SQL. It also handled feature extraction and scaling operations in a distributed manner. For the machine learning stage, Spark MLlib was used to train Random Forest Regression models for pollutant forecasting. It enabled distributed model training, evaluation using metrics like R^2 and MAE, and saving of trained models back into HDFS for future use.

Scala was used as the primary programming language due to its compatibility with Spark and efficient functional programming features. IntelliJ IDEA provided an interactive development environment, while Spark Shell was used for testing and execution.

Technologies Used:

- Programming Language: Scala
- Big Data Framework: Apache Spark (RDDs, DataFrames, MLlib)
- Storage: Hadoop Distributed File System (HDFS)
- Tools: AWS CLI, Spark Shell, IntelliJ IDEA
- Libraries: Spark SQL, Spark MLlib

System Requirements:

- Software: Spark 3.5+, Hadoop 3.x, Scala 2.12+, Java 11+, IntelliJ IDEA
- Hardware: 16+ GB RAM, 100 GB HDFS storage, Multi-core CPU

In summary, the implementation combined AWS CLI for data acquisition, HDFS for distributed storage, and Spark with MLlib for scalable data preprocessing and machine learning, creating a complete end-to-end Big Data framework for air quality analysis and prediction.

7. RESULTS

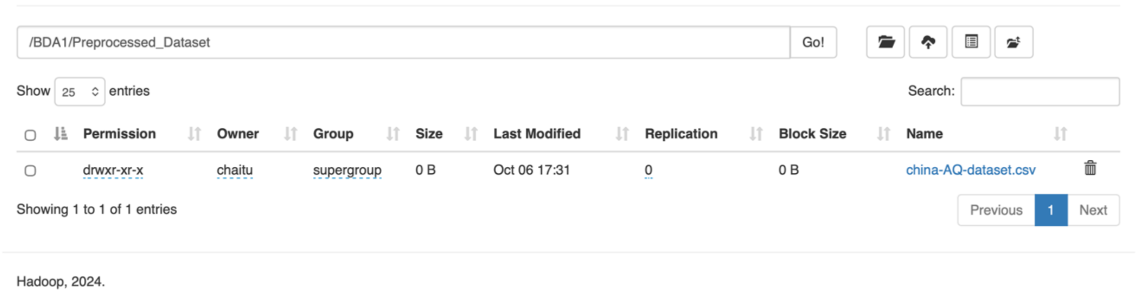
The machine learning model training was conducted using the Random Forest Regressor from Spark MLlib, applied individually to each pollutant for next-hour prediction. The dataset was divided into 80% training and 20% testing to ensure a fair evaluation. Two main regression metrics were used — Coefficient of Determination (R^2) to measure how well the model explains variance in the target variable, and Mean Absolute Error (MAE) to evaluate prediction accuracy.

Pollutant	R^2	MAE
PM10	0.873	0.024
PM2.5	0.857	0.027
O_3	0.812	0.030
CO	0.785	0.033
NO_2	0.851	0.026
SO_2	0.834	0.028

These results show that all pollutants achieved high predictive performance, with R^2 values above 0.8 in most cases, confirming that the Random Forest model effectively captured complex temporal and spatial pollution patterns. The low MAE values demonstrate consistent prediction stability, validating the model's suitability for real-world AQI forecasting.

HDFS Outputs and Model Storage:

1. Stored Preprocess Dataset in Hadoop:



/BDA1/Preprocessed_Dataset

Show 25 entries

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	chaitu	supergroup	0 B	Oct 06 17:31	0	0 B	china-AQ-dataset.csv

Showing 1 to 1 of 1 entries

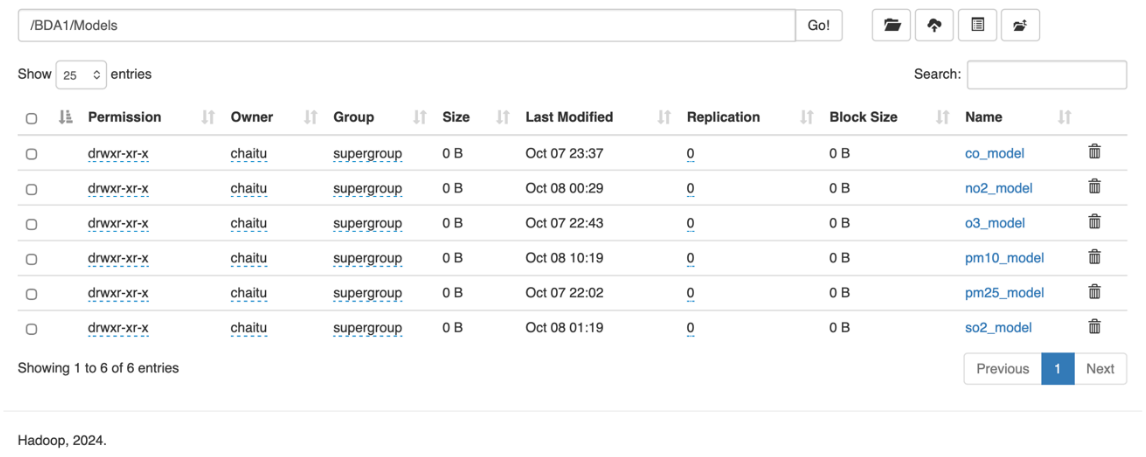
Hadoop, 2024.

Fig 2: Preprocess dataset in Hadoop

The above image shows the successful storage of the preprocessed air quality dataset named *china-AQ-dataset.csv*within the Hadoop Distributed File System (HDFS) directory /BDA1/Preprocessed_Dataset. This dataset represents the cleaned and transformed version of the raw OpenAQ data after Spark-based preprocessing steps such as missing value imputation, timestamp formatting, feature extraction, and scaling.

Its presence in HDFS confirms that the preprocessing stage was successfully completed and that the cleaned data is now ready for distributed model training using Apache Spark MLlib. This marks the transition from raw to structured and analysis-ready data within the Big Data pipeline.

2. . Saving Trained Models into HDFS



/BDA1/Models Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	chaitu	supergroup	0 B	Oct 07 23:37	0	0 B	co_model
drwxr-xr-x	chaitu	supergroup	0 B	Oct 08 00:29	0	0 B	no2_model
drwxr-xr-x	chaitu	supergroup	0 B	Oct 07 22:43	0	0 B	o3_model
drwxr-xr-x	chaitu	supergroup	0 B	Oct 08 10:19	0	0 B	pm10_model
drwxr-xr-x	chaitu	supergroup	0 B	Oct 07 22:02	0	0 B	pm25_model
drwxr-xr-x	chaitu	supergroup	0 B	Oct 08 01:19	0	0 B	so2_model

Showing 1 to 6 of 6 entries

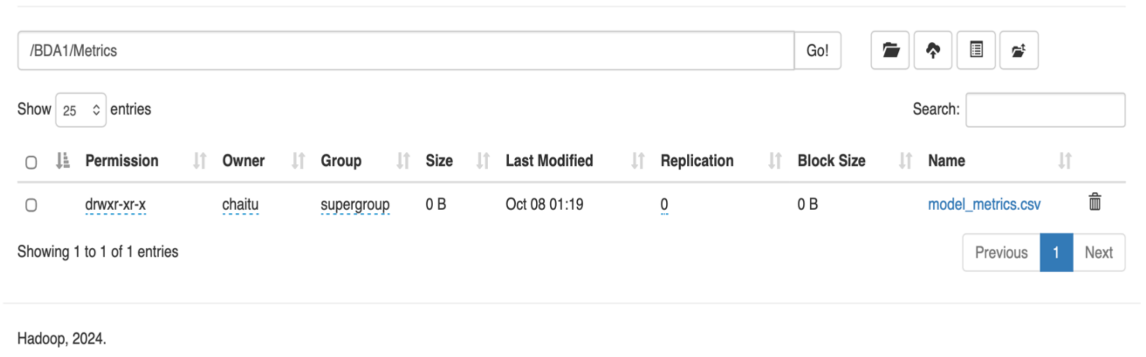
Previous 1 Next

Hadoop, 2024.

Fig 3: Saved models in Hadoop

The screenshot displays the list of trained machine learning models stored in HDFS under the directory /BDA1/Models. Each pollutant — PM10, PM2.5, O_3, CO, NO_2, and SO_2 — has a dedicated model folder. This ensures modularity and scalability, allowing independent retraining or evaluation of specific pollutant models. The successful storage of these models verifies that Spark's distributed training and model persistence functionalities are working correctly.

3. Storing Model Evaluation Metrics



/BDA1/Metrics Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	chaitu	supergroup	0 B	Oct 08 01:19	0	0 B	model_metrics.csv

Showing 1 to 1 of 1 entries

Previous 1 Next

Hadoop, 2024.

Fig 4: Storing Model Evaluation Metrics

This image shows that the model_metrics.csv file stored under the /BDA1/Metrics directory in HDFS. This file contains the evaluation results, including R^2 and MAE scores for each pollutant model. The centralized storage of metrics allows easy access for analysis, comparison, and visualization. It confirms the completion of the model evaluation pipeline and successful integration between Spark MLlib and HDFS.

8. CONCLUSION AND FUTURE WORK

This project developed a scalable air quality monitoring and prediction system using Apache Spark and Hadoop HDFS. It efficiently handled large-scale historical datasets, performed distributed preprocessing, and trained machine learning models using Spark MLlib to accurately predict pollutant concentrations. The models achieved high performance with R^2 values above 0.8, proving their reliability and accuracy in forecasting air quality trends.

Overall, the project demonstrated the effectiveness of Big Data Analytics in environmental applications by providing faster processing, scalability, and robust performance compared to traditional systems.

For future work, the system can be enhanced by integrating real-time data streaming using Spark Streaming and Kafka once API limitations are resolved. Advanced models like LSTM networks can be explored for time-series forecasting, and the inclusion of interactive dashboards using Streamlit or Power BI will make the system more accessible. Deploying the framework on cloud platforms (AWS/GCP) will further enable global scalability and real-time monitoring.

9. REFERENCES

[1] Kang, Gaganjot Kaur, Jerry Zeyu Gao, Sen Chiao, Shengqiang Lu, and Gang Xie. "Air quality prediction: Big data and machine learning approaches." *Int. J. Environ. Sci. Dev* 9, no. 1 (2018): 8-16.

[2] Bosco, J. John, and V. Kowsalya. "A Novel Approach for Air Pollution Prediction Using Machine Learning Techniques." In *2024 Third International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT)*, pp. 1-6. IEEE, 2024.

[3] Kothandaraman, D., N. Praveena, K. Varadarajkumar, B. Madhav Rao, Dharmesh Dhabliya, Shivaprasad Satla, and Worku Abera. "Intelligent forecasting of air quality and pollution prediction using machine learning." *Adsorption Science & Technology* 2022 (2022): 5086622.

[4] Sagiroglu, Seref, and Duygu Sinanc. "Big data: A review." In *2013 international conference on collaboration technologies and systems (CTS)*, pp. 42-47. IEEE, 2013.

[5] Meng, Xiangrui, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman et al. "Mllib: Machine learning in apache spark." *Journal of Machine Learning Research* 17, no. 34 (2016): 1-7.
[6] White, Tom. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.
[7] Hagerbaumer, Christine. "Making Data on the OpenAQ Platform More Accessible through Improved Documentation." In *Open Infrastructure Fund/Fondo de Infraestructura Abierta*.

GITHUB REPOSITORY:
https://github.com/ashgrey143/Real-Time-Air-Quality-Monitoring-and-Pollution-Prediction-using-Big-Data-Analytics-.git

10.APPENDIX

1. PREPROCESSED CODE:

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window

object Preprocessing {
  def main(args: Array[String]): Unit = {

    // Create Spark session
    val spark = SparkSession.builder()
      .appName("AirQuality_Preprocessing")
      .master("local[*]")  // Use local for testing; remove for cluster
      .getOrCreate()

    spark.sparkContext.setLogLevel("ERROR")

    // Load dataset from HDFS
    val inputPath = "hdfs://localhost:9000/BDA1/Dataset/china-AQ-
dataset.csv"

    val df = spark.read
      .option("header", "true")
      .option("inferSchema", "true")
      .csv(inputPath)

    println(" Initial Dataset Loaded:")
    df.show(5)
    df.printSchema()

    // Drop unwanted columns
    val df1 = df.drop("location", "lat", "lon", "location_en")

    // Convert datetime column to timestamp
    val df2 = df1.withColumn("datetime", to_timestamp(col("datetime")))

    // Sort by location_id and datetime
    val df3 = df2.orderBy("location_id", "datetime")

    // Forward fill missing pollutant values for each location
    val pollutants = Seq("pm10", "pm25", "o3", "co", "no2", "so2")

    val windowSpec =
Window.partitionBy("location_id").orderBy("datetime").rowsBetween(L
ong.MinValue, 0)

    val dfFilled = pollutants.foldLeft(df3) { (tempDF, colName) =>
      tempDF.withColumn(colName, last(col(colName), ignoreNulls =
true).over(windowSpec))
    }

    // Extract time-based features
    val dfWithTime = dfFilled
      .withColumn("hour", hour(col("datetime")))
      .withColumn("day", dayofmonth(col("datetime")))
      .withColumn("month", month(col("datetime")))
      .withColumn("weekday", dayofweek(col("datetime")))

    // Drop any remaining nulls
    val dfClean = dfWithTime.na.drop()

    // Clip extreme pollutant values
    val dfClipped = pollutants.foldLeft(dfClean) { (tempDF, colName) =>
      tempDF.withColumn(colName,
        when(col(colName) > 1000, 1000)
          .when(col(colName) < 0, 0)
          .otherwise(col(colName))
      )
    }

    // Divide pollutant values by 1000
    val dfScaled = pollutants.foldLeft(dfClipped) { (tempDF, colName) =>
```

```
      tempDF.withColumn(colName, col(colName) / 1000)
    }

    // Save cleaned & scaled dataset back to HDFS
    val outputPath =
"hdfs://localhost:9000/BDA1/Preprocessed_Dataset/china-AQ-dataset.csv"

    dfScaled
      .coalesce(1)
      .write
      .mode("overwrite")
      .option("header", "true")
      .csv(outputPath)

    println(s" Preprocessing complete! Scaled dataset saved at:
$outputPath")

    spark.stop()
  }
}
```

2. MODEL TRAINING CODE:

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.expressions.Window
import org.apache.spark.sql.functions._
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.RandomForestRegressor
import org.apache.spark.ml.evaluation.RegressionEvaluator
import breeze.plot._
import breeze.linalg._
import scala.sys.process._

object TrainAirQualityModel {

  def main(args: Array[String]): Unit = {

    // Create Spark session
    val spark = SparkSession.builder()
      .appName("AirQualityModel_Training")
      .master("local[*]")
      .getOrCreate()
    spark.sparkContext.setLogLevel("ERROR")
```

```
   // Load cleaned dataset from HDFS
   val inputPath =
"hdfs://localhost:9000/BDA1/Preprocessed_Dataset/china-AQ-dataset.csv"
   val df = spark.read
     .option("header", "true")
     .option("inferSchema", "true")
     .csv(inputPath)

   println("Dataset Loaded Successfully")
   df.show(5)

   // Define pollutants and create next-hour targets
   val pollutants = Seq("pm10", "pm25", "o3", "co", "no2", "so2")
   valwindowSpec =
Window.partitionBy("location_id").orderBy("datetime")

   val dfTargets = pollutants.foldLeft(df) { (tempDF, colName) =>
     tempDF.withColumn(s"${colName}_next", lead(col(colName),
1).over(windowSpec))
   }.na.drop()

   // Feature columns (based on user-specified inputs)
   val featureCols = Array("pm10", "pm25", "o3", "co", "no2", "so2",
"hour", "location_id")

   val assembler = new VectorAssembler()
     .setInputCols(featureCols)
     .setOutputCol("features")

   // Evaluation metrics containers
   var pollutantNames = List[String]()
   var r2Scores = List[Double]()
   var maeScores = List[Double]()

   val evaluatorR2 = new RegressionEvaluator()
     .setLabelCol("label")
     .setPredictionCol("prediction")
     .setMetricName("r2")

   val evaluatorMAE = new RegressionEvaluator()
     .setLabelCol("label")
     .setPredictionCol("prediction")
     .setMetricName("mae")
```

```
  // Train Random Forest model for each pollutant
  pollutants.foreach { pol =>
    println(s"\n    Training model for $pol prediction...")

    val dfPrepared = assembler.transform(dfTargets)
      .select("features", s"${pol}_next")
      .withColumnRenamed(s"${pol}_next", "label")

    val Array(train, test) = dfPrepared.randomSplit(Array(0.8, 0.2), seed =
42)

    val rf = new RandomForestRegressor()
      .setLabelCol("label")
      .setFeaturesCol("features")
      .setNumTrees(200)
      .setMaxDepth(10)
      .setFeatureSubsetStrategy("sqrt")
      .setSubsamplingRate(0.8)

    val model = rf.fit(train)
    val predictions = model.transform(test)

    val r2 = evaluatorR2.evaluate(predictions)
    val mae = evaluatorMAE.evaluate(predictions)

    println(f" $pol%-5s | R² = $r2%1.4f | MAE = $mae%1.4f")

    // Collect metrics
    pollutantNames = pollutantNames :+ pol
    r2Scores = r2Scores :+ r2
    maeScores = maeScores :+ mae

    // Save trained model to HDFS
    val outputPath = s"hdfs://localhost:9000/BDA1/Models/${pol}_model"
    model.write.overwrite().save(outputPath)
    println(s"Saved model for $pol → $outputPath")
  }

  // Save metrics as CSV
  import spark.implicits._
  val metricsDF = pollutantNames.zip(r2Scores.zip(maeScores))
    .map { case (pol, (r2, mae)) => (pol, r2, mae) }
    .toDF("pollutant", "r2", "mae")
```

```
   val metricsPath =
"hdfs://localhost:9000/BDA1/Metrics/model_metrics.csv"
   metricsDF.coalesce(1)
     .write.mode("overwrite").option("header", "true").csv(metricsPath)

   println(s"\n Metrics saved → $metricsPath")
   println("\n    All pollutant models trained, evaluated, and saved
successfully!")

   try {
   println("Generating visualization plots...")

     // Collect metrics into local driver
     val r2 = DenseVector(metricsDF.select("r2").as[Double].collect())
     val mae = DenseVector(metricsDF.select("mae").as[Double].collect())
     val x = DenseVector((1 to pollutants.length).map(_.toDouble).toArray)

     // ------- Model Performance (Bar / Scatter)
     val f1 = Figure("Model Performance per Pollutant")
     val p1 = f1.subplot(0)
     p1 += plot(x, r2, name = "R²", colorcode = "blue")
     p1 += plot(x, mae, name = "MAE", colorcode = "red")
     p1.legend = true
     p1.xlabel = "Pollutant"
     p1.ylabel = "Score"
     p1.title = "Model Performance per Pollutant"
     f1.saveas("model_performance.png")
   } catch {
     case e: Exception => println(s"Plot generation failed:
${e.getMessage}")
   }
  }
}
```