SAT5165 – Big Data Analytics

Large Project Report

Project Title: **Large-Scale Analysis of Air Quality and Respiratory Health Risks Using PySpark**

Team Members: Mary Nnipaa Meteku, Dennis Owusu, Uttam Kumar Bellamkonda, Sucharitha Reddy Dammareddygari, Fredrick Damptey

GitHub Repository: https://github.com/RoyalDennis

1. Introduction

This project explores large-scale air quality data using Apache Spark for distributed processing. It examines pollutant trends and their impact on air quality, employing Spark MLlib for statistical and predictive analysis. The dataset used was the U.S. Pollution Data (2000–2016), with over 2.5 million records of PM2.5, Ozone, SO_2, and NO_2 levels. The goal was to clean, process, and analyze the data efficiently while testing Spark's performance on multiple virtual machines.

2. Environment Configuration

Each member configured a Spark cluster on two virtual machines. The IP configurations are listed below:

Team	VM1 (Master)	VM2 (Worker)
Mary Nnipaa Meteku	192.168.13.146	192.168.13.147
Dennis Owusu	192.168.13.179	192.168.13.180
Uttam Kumar Bellamkonda	192.168.13.107	192.168.13.108
Sucharitha Reddy Dammareddygari	192.168.13.113	192.168.13.114
Fredrick Damptey	192.168.13.116	192.168.13.117

3. Explanation of Codes and Method

The PySpark project begins by loading the U.S. Pollution dataset with read.csv(), using inferred headers and schema to create a Spark DataFrame. Column names are cleaned to remove spaces and special characters for easier referencing. Missing numeric values are replaced with column means using pyspark.sql.functions.mean(), while missing text entries (State, County, City) are filled with "Unknown." A loop checks for remaining nulls, and printSchema() confirms correct data types. The cleaned data is saved with write.csv() in overwrite mode. Descriptive statistics are produced with describe(), and groupBy() with avg() computes average pollutant levels by state. Correlation analysis is performed using MLlib's VectorAssembler and Correlation.corr() to calculate Pearson correlations between pollutants and the Air Quality Index (AQI). Finally, Spark cluster commands (start-master.sh, start-worker.sh) manage master–worker connections, and the Spark Web UI monitors cluster activity and job progress.

4.1 Mary Nnipaa Meteku – Data Cleaning and Missing Value Handling

I focused on cleaning and preparing the dataset for analysis. I used PySpark DataFrames to read, rename columns, impute missing values, check schema consistency, and export the cleaned data. To start, I set up and verified the Spark master and worker nodes across his virtual machines (VMs: 192.168.13.146 and 192.168.13.147), ensuring that distributed computation was operational for scalable data analysis.

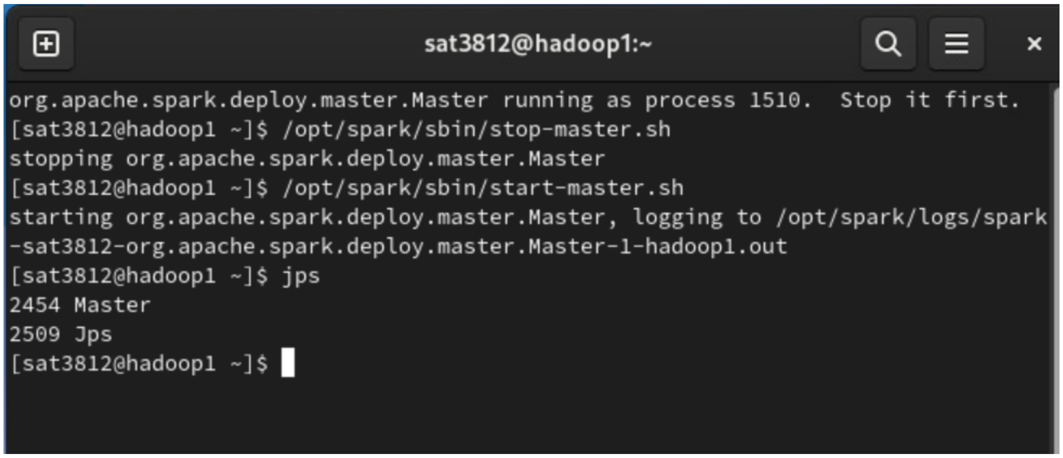


Figure 1.1: Mary's Spark Master Node (192.168.13.146) started successfully.

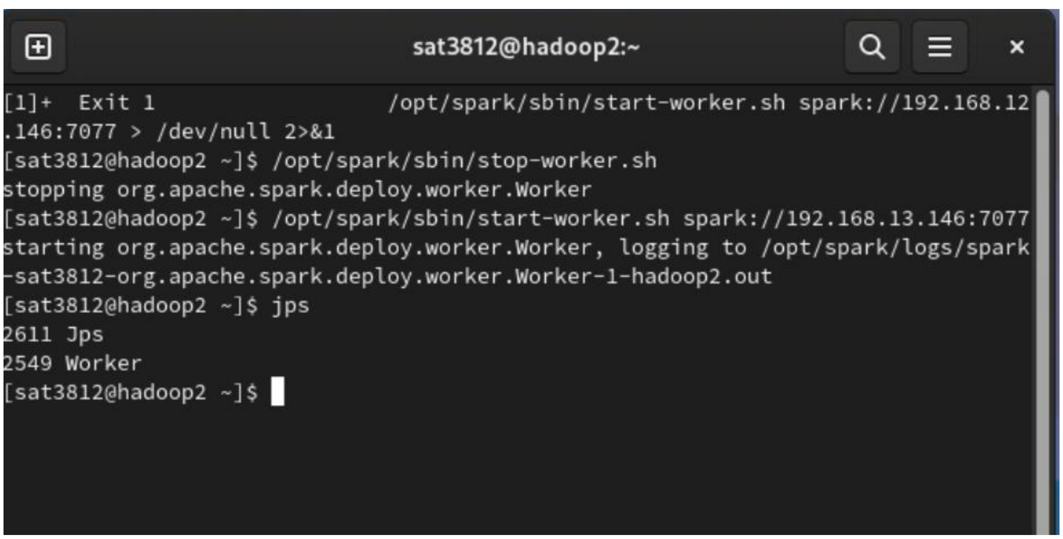


Figure 1.2: Mary's Worker Node (192.168.13.147) connected to Spark Master.

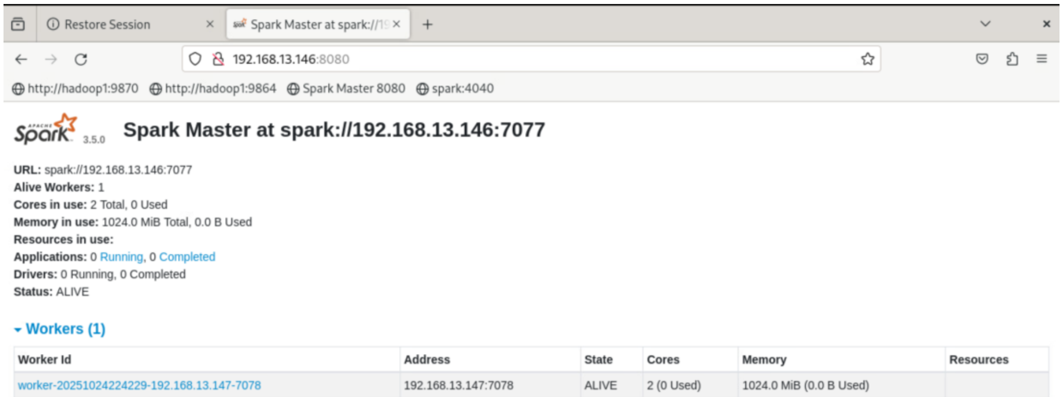
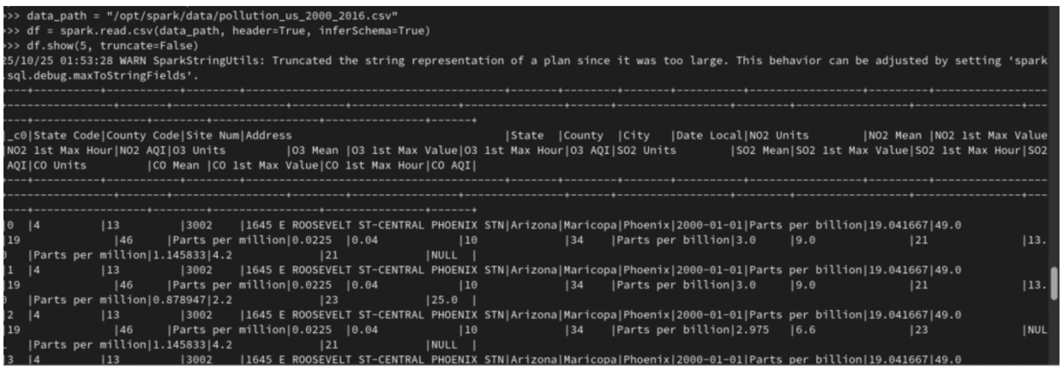


Figure 1.3: Mary's Spark Web UI showing connected worker node.



```
>>> data_path = "/opt/spark/data/pollution_us_2000_2016.csv"
>>> df = spark.read.csv(data_path, header=True, inferSchema=True)
>>> df.show(5, truncate=False)
25/10/25 01:53:28 WARN SparkStringUtils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark
.sql.debug.maxToStringFields'.
|_c0|State Code|County Code|Site Num|Address                               |State  |County  |City   |Date Local|NO2 Units        |NO2 Mean |NO2 1st Max Value
|NO2 1st Max Hour|NO2 AQI|O3 Units         |O3 Mean |O3 1st Max Value|O3 1st Max Hour|O3 AQI|SO2 Units        |SO2 Mean|SO2 1st Max Value|SO2 1st Max Hour|SO2
 AQI|CO Units         |CO Mean |CO 1st Max Value|CO 1st Max Hour|CO AQI|
|0  |4         |13         |3002    |1645 E ROOSEVELT ST-CENTRAL PHOENIX STN|Arizona|Maricopa|Phoenix|2000-01-01|Parts per billion|19.041667|49.0
|19              |46     |Parts per million|0.0225  |0.04            |10             |34    |Parts per billion|3.0     |9.0              |21              |13.
0  |Parts per million|1.145833|4.2             |21             |NULL  |
|1  |4         |13         |3002    |1645 E ROOSEVELT ST-CENTRAL PHOENIX STN|Arizona|Maricopa|Phoenix|2000-01-01|Parts per billion|19.041667|49.0
|19              |46     |Parts per million|0.0225  |0.04            |10             |34    |Parts per billion|3.0     |9.0              |21              |13.
0  |Parts per million|0.878947|2.2             |23             |25.0  |
|2  |4         |13         |3002    |1645 E ROOSEVELT ST-CENTRAL PHOENIX STN|Arizona|Maricopa|Phoenix|2000-01-01|Parts per billion|19.041667|49.0
|19              |46     |Parts per million|0.0225  |0.04            |10             |34    |Parts per billion|2.975   |6.6              |23              |NUL
L  |Parts per million|1.145833|4.2             |21             |NULL  |
|3  |4         |13         |3002    |1645 E ROOSEVELT ST-CENTRAL PHOENIX STN|Arizona|Maricopa|Phoenix|2000-01-01|Parts per billion|19.041667|49.0
```

Figure 1.4: Initial data loading and preview of the dataset.

```
>>> for old_col in df.columns:
...     new_col = (
...             old_col.strip()
...             .replace(" ","_")
...             .replace("/","_")
...             .replace(".","_")
...             .replace("(","")
...             .replace(")","")
...     )
...     df = df.withColumnRenamed(old_col, new_col)
...
>>> df.columns
['_c0', 'State_Code', 'County_Code', 'Site_Num', 'Address', 'State', 'County', 'City', 'Date_Local', 'NO2_Units', 'NO2_Mean', 'NO2_1st_Max_Value', 'NO2_1st_Ma
x_Hour', 'NO2_AQI', 'O3_Units', 'O3_Mean', 'O3_1st_Max_Value', 'O3_1st_Max_Hour', 'O3_AQI', 'SO2_Units', 'SO2_Mean', 'SO2_1st_Max_Value', 'SO2_1st_Max_Hour',
'SO2_AQI', 'CO_Units', 'CO_Mean', 'CO_1st_Max_Value', 'CO_1st_Max_Hour', 'CO_AQI']
>>> from pyspark.sql.function import col, mean
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'pyspark.sql.function'
>>> from pyspark.sql.functions import col, mean
>>> numeric_col = ["NO2_Mean", "O3_Mean", "SO2_Mean", "CO_Mean"]
>>> numeric_cols = ["NO2_Mean", "O3_Mean", "SO2_Mean", "CO_Mean"]
>>> for col_name in numeric_cols:
...     mean_value = df.select(mean(col(col_name))).collect()[0][0]
```

Figure 1.5: Column renaming and missing value handling.

```
...
>>> df = df.na.fill({
...     "State": "Unknown",
...     "County": "UNknown",
...     "City": "Unknown",
...     "Address": "Unknown"
... })
>>> from pyspark.sql.functions import col, sum
>>> check_cols = ["NO2_Mean", "O3_Mean", "SO2_Mean", "CO_Mean", "State", "City"]
>>> for c in check_cols:
...     null_count = df.select(sum(col(c).isNull().cast("int")).alias(c)).first()[0]
...     print(f"{c}: {null_count}")
...
NO2_Mean: 0
O3_Mean: 0
SO2_Mean: 0
CO_Mean: 0
State: 0
City: 0
>>> print(f"Total Rows: {df.count()}")
Total Rows: 1746661
>>> print(f"Total Columns: {len(df.columns)}")
```

Figure 1.6: Verification of null value counts after imputation.

```
>>> df.printSchema()
root
 |-- _c0: integer (nullable = true)
 |-- State_Code: integer (nullable = true)
 |-- County_Code: integer (nullable = true)
 |-- Site_Num: integer (nullable = true)
 |-- Address: string (nullable = false)
 |-- State: string (nullable = false)
 |-- County: string (nullable = false)
 |-- City: string (nullable = false)
 |-- Date_Local: date (nullable = true)
 |-- NO2_Units: string (nullable = true)
 |-- NO2_Mean: double (nullable = false)
 |-- NO2_1st_Max_Value: double (nullable = true)
 |-- NO2_1st_Max_Hour: integer (nullable = true)
 |-- NO2_AQI: integer (nullable = true)
 |-- O3_Units: string (nullable = true)
 |-- O3_Mean: double (nullable = false)
 |-- O3_1st_Max_Value: double (nullable = true)
 |-- O3_1st_Max_Hour: integer (nullable = true)
 |-- O3_AQI: integer (nullable = true)
 |-- SO2_Units: string (nullable = true)
 |-- SO2_Mean: double (nullable = false)
 |-- SO2_1st_Max_Value: double (nullable = true)
 |-- SO2_1st_Max_Hour: integer (nullable = true)
```

Figure 1.7: Schema inspection after cleaning.

```
only showing top 10 rows

>>> df.write.csv("/opt/spark/data/pollution_cleaned.csv", header=True, mode="overwrite")
>>> summary_cols = ["NO2_Mean", "O3_Mean", "SO2_Mean, "CO_Mean"]
  File "<stdin>", line 1
    summary_cols = ["NO2_Mean", "O3_Mean", "SO2_Mean, "CO_Mean"]
                                                               ^
SyntaxError: unterminated string literal (detected at line 1)
>>> summary_cols = ["NO2_Mean", "O3_Mean", "SO2_Mean", "CO_Mean"]
>>> df.select(summary_cols).describe().show(truncate=False)
+-------+-----------------+-------------------+------------------+-------------------+
|summary|NO2_Mean         |O3_Mean            |SO2_Mean          |CO_Mean            |
+-------+-----------------+-------------------+------------------+-------------------+
```

Figure 1.8: Cleaned dataset successfully saved in Spark directory.

3.2 Dennis Owusu – Correlation Analysis

My section involved performing correlation analysis using Spark MLlib. I calculated Pearson correlations between NO_2, SO_2, O_3, CO, and AQI, then displayed correlation matrices and schema outputs. To start, I set up and verified the Spark master and worker nodes across his virtual machines (VMs: 192.168.13.179 and 192.168.13.180), ensuring that distributed computation was operational for scalable data analysis.

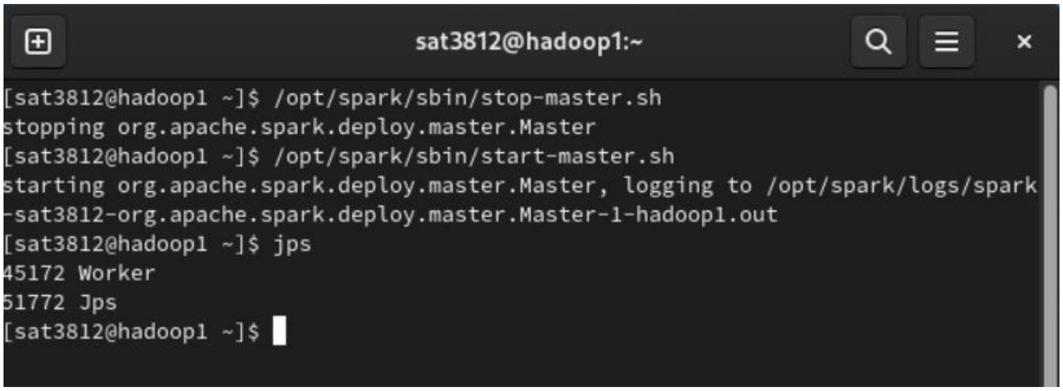


Figure 2.1: Dennis's Spark Master Node (192.168.13.179) started successfully.

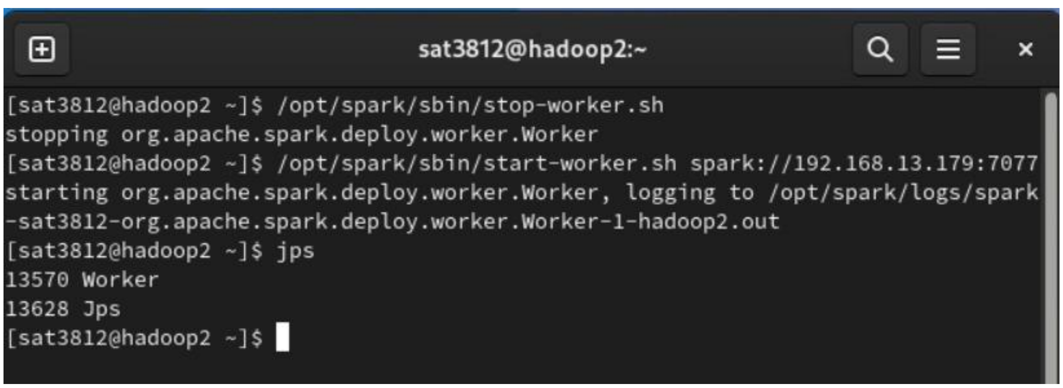


Figure 2.2: Dennis's Worker Node (192.168.13.180) connected to Master.

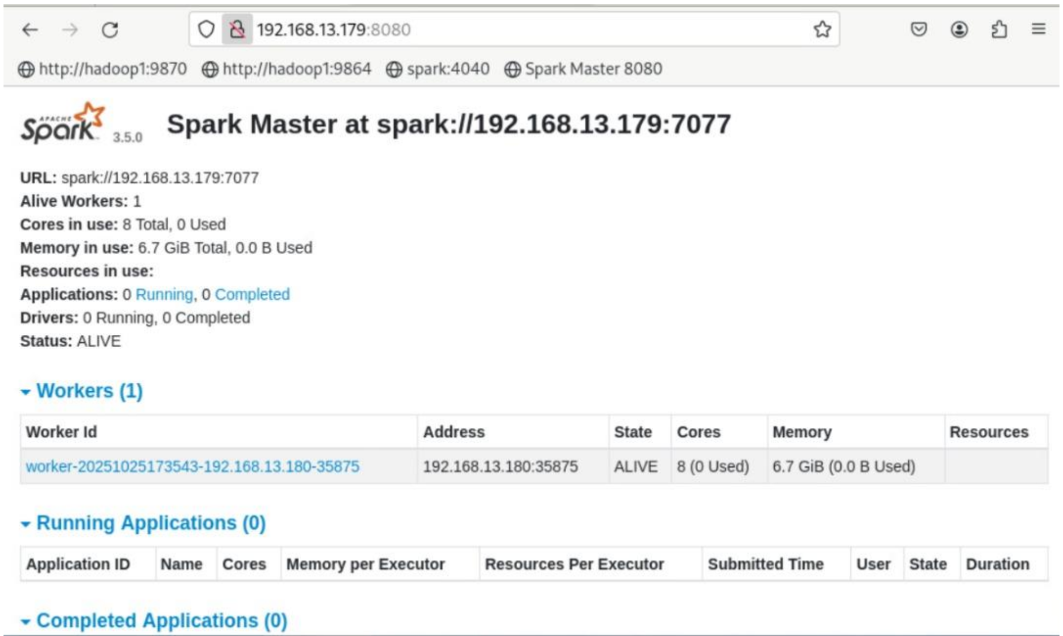


Figure 2.3: Spark Master Web UI (Dennis) confirming one active worker node.

```
>>> df.count
<bound method DataFrame.count of DataFrame[_c0: int, State_Code: int, County_Cod
e: int, Site_Num: int, Address: string, State: string, County: string, City: str
ing, Date_Local: date, NO2_Units: string, NO2_Mean: double, NO2_1st_Max_Value: d
ouble, NO2_1st_Max_Hour: int, NO2_AQI: int, O3_Units: string, O3_Mean: double, O
3_1st_Max_Value: double, O3_1st_Max_Hour: int, O3_AQI: int, SO2_Units: string, S
O2_Mean: double, SO2_1st_Max_Value: double, SO2_1st_Max_Hour: int, SO2_AQI: doub
le, CO_Units: string, CO_Mean: double, CO_1st_Max_Value: double, CO_1st_Max_Hour
: int, CO_AQI: double]>
>>> 
```

Figure 2.4: DataFrame count verification before correlation analysis.

```
>>> spark.stop()
>>> from pyspark.sql import SparkSession
>>> spark = (
...     SparkSession.builder
...     .appName("Air Quality Corr")
...     .master("local[*]")
...     .getOrCreate()
... )
>>> df = (spark.read
...     .option("header", "true")
...     .option("inferSchema", "true")
...     .csv("file:///home/sat3812/pollution_cleaned/"))
>>> df.show(5)
25/10/26 15:38:36 WARN SparkStringUtils: Truncated the string representation of
a plan since it was too large. This behavior can be adjusted by setting 'spark.s
ql.debug.maxToStringFields'.
+-----+----------+-----------+--------+--------------------+------+---------+---
--------+----------+----------------+--------+-----------------+-------------
-+-------+----------------+--------+----------------+---------------+------+---
-------------+--------+----------------+-----------------+-------+------------
----+--------+----------------+---------------+------+
|  _c0|State_Code|County_Code|Site_Num|             Address| State|   County|
   City|Date_Local|        NO2_Units|NO2_Mean|NO2_1st_Max_Value|NO2_1st_Max_Hou
r|NO2_AQI|          O3_Units| O3_Mean|O3_1st_Max_Value|O3_1st_Max_Hour|O3_AQI|
```

Figure 2.5: Cleaned dataset loaded for correlation analysis.

```
----+--------+---------------+-------------+------+
|  _c0|State_Code|County_Code|Site_Num|            Address| State|   County|
    City|Date_Local|        NO2_Units|NO2_Mean|NO2_1st_Max_Value|NO2_1st_Max_Hou
r|NO2_AQI|         O3_Units| O3_Mean|O3_1st_Max_Value|O3_1st_Max_Hour|O3_AQI|
     SO2_Units|SO2_Mean|SO2_1st_Max_Value|SO2_1st_Max_Hour|SO2_AQI|         CO_U
nits| CO_Mean|CO_1st_Max_Value|CO_1st_Max_Hour|CO_AQI|
+-----+----------+-----------+--------+-------------------+------+---------+---
--------+----------+----------------+--------+-----------------+--------------
-+-------+----------------+--------+----------------+---------------+------+---
-------------+--------+-----------------+----------------+-------+------------
----+--------+---------------+-------------+------+
|50487|        20|        209|      21|1210 N. 10TH ST.,...|Kansas|Wyandotte|Kan
sas City|2006-04-16|Parts per billion|4.541667|             13.0|
2|     12|Parts per million|0.038458|          0.057|              9|    48|Par
ts per billion|0.958333|             2.0|               2|    3.0|Parts per mil
lion|   0.025|             0.1|              4|  NULL|
|50488|        20|        209|      21|1210 N. 10TH ST.,...|Kansas|Wyandotte|Kan
sas City|2006-04-16|Parts per billion|4.541667|             13.0|
2|     12|Parts per million|0.038458|          0.057|              9|    48|Par
ts per billion|0.958333|             2.0|               2|    3.0|Parts per mil
lion|0.020833|             0.1|              0|   1.0|
|50489|        20|        209|      21|1210 N. 10TH ST.,...|Kansas|Wyandotte|Kan
sas City|2006-04-16|Parts per billion|4.541667|             13.0|
2|     12|Parts per million|0.038458|          0.057|              9|    48|Par
```

Figure 2.6: Sample data displayed before correlation computation.

```
>>> df.printSchema()
root
 |-- _c0: integer (nullable = true)
 |-- State_Code: integer (nullable = true)
 |-- County_Code: integer (nullable = true)
 |-- Site_Num: integer (nullable = true)
 |-- Address: string (nullable = true)
 |-- State: string (nullable = true)
 |-- County: string (nullable = true)
 |-- City: string (nullable = true)
 |-- Date_Local: date (nullable = true)
 |-- NO2_Units: string (nullable = true)
 |-- NO2_Mean: double (nullable = true)
 |-- NO2_1st_Max_Value: double (nullable = true)
 |-- NO2_1st_Max_Hour: integer (nullable = true)
 |-- NO2_AQI: integer (nullable = true)
 |-- O3_Units: string (nullable = true)
 |-- O3_Mean: double (nullable = true)
 |-- O3_1st_Max_Value: double (nullable = true)
 |-- O3_1st_Max_Hour: integer (nullable = true)
 |-- O3_AQI: integer (nullable = true)
 |-- SO2_Units: string (nullable = true)
 |-- SO2_Mean: double (nullable = true)
```

Figure 2.7: Schema validation confirming dataset readiness for correlation.

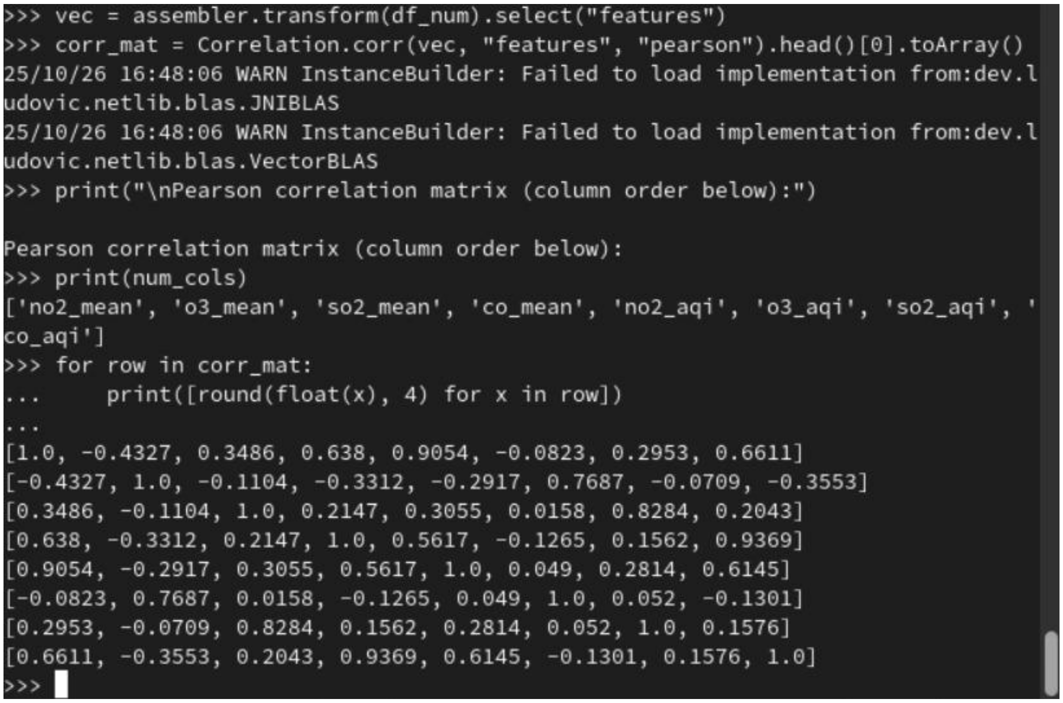


Figure 2.8: Pearson correlation matrix generated using Spark MLlib.

3.3 Uttam Kumar Bellamkonda – Regression Analysis

The goal of the regression analysis is to predict the Air Quality Index (AQI) based on pollutant concentration variables—NO_2, O_3, SO_2, CO, PM2.5, and PM10, using a linear regression model in PySpark. To start, I set up and verified the Spark master and worker nodes across his virtual machines (VMs: 192.168.13.107 and 192.168.13.108), ensuring that distributed computation was operational for scalable data analysis.

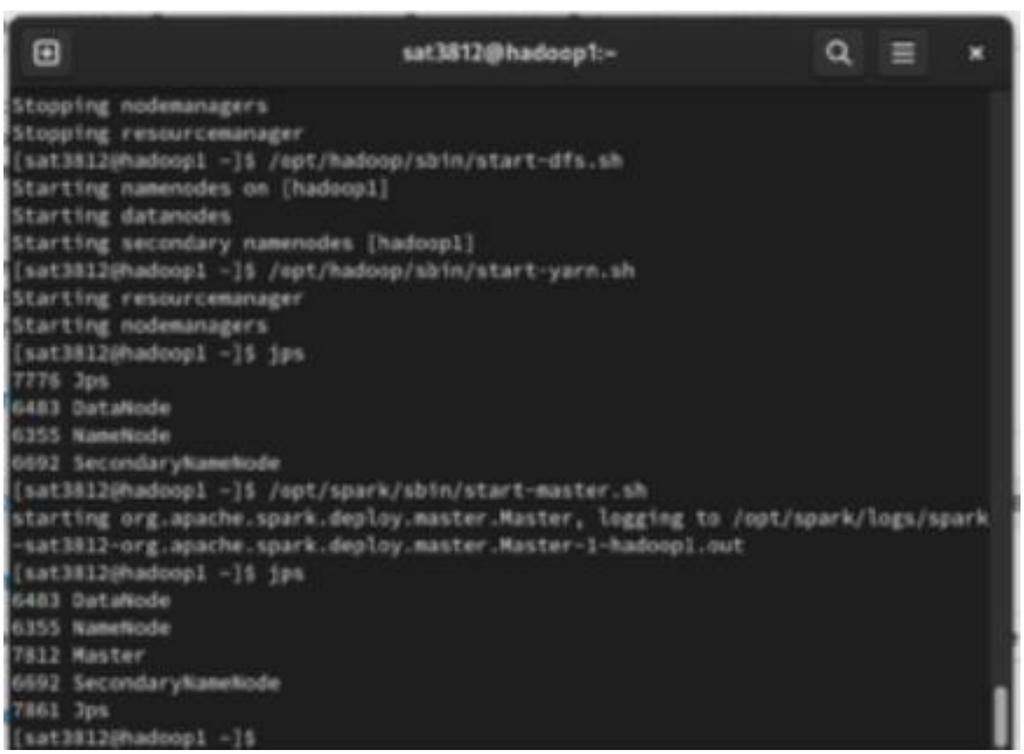


Figure 3.1: Uttam’s Spark Master Node (192.168.13.107) started successfully.

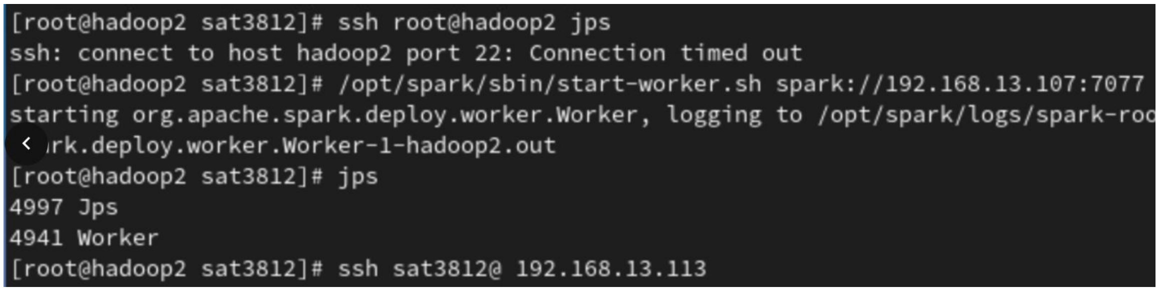


Figure 3.2: Uttam’s Worker Node (192.168.13.108) connected to Spark Master.

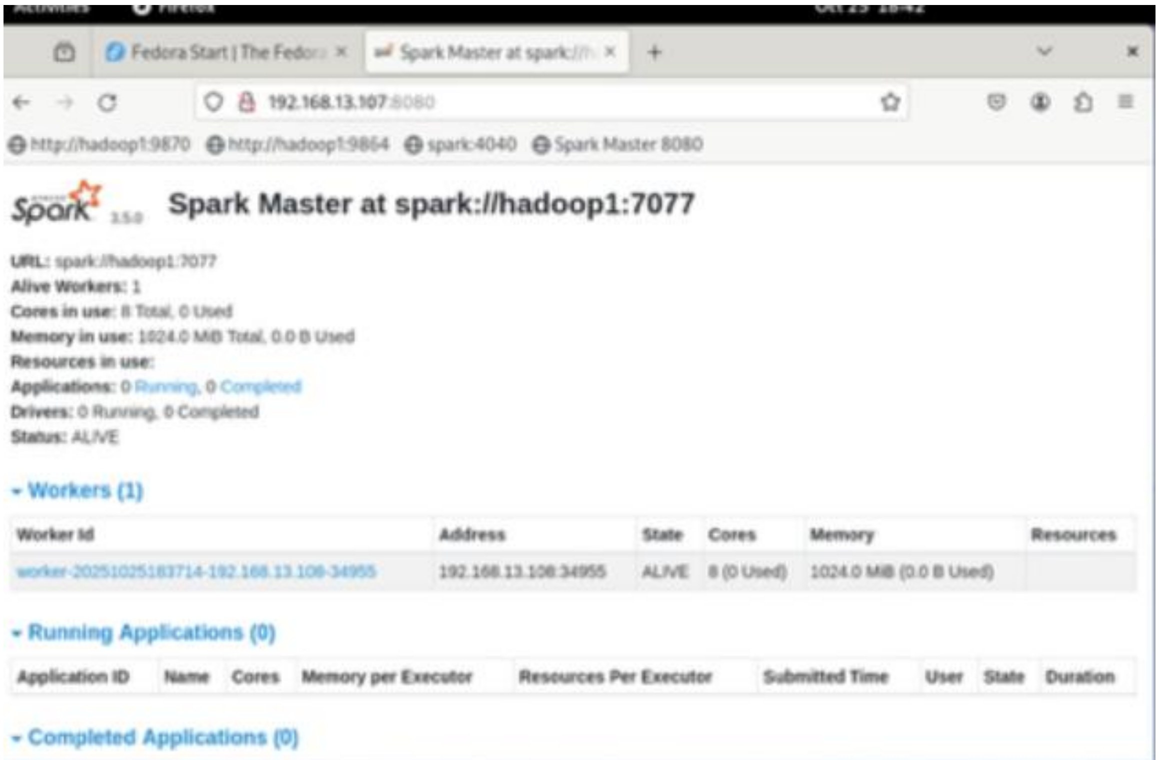


Figure 3.3: Spark Master Web UI (Uttam) confirming one active worker node

The dataset is loaded using the spark.read.csv() function with headers and schema inference enabled. The code dynamically identifies pollutant columns (NO_2_AQI, O_3_AQI, SO_2_AQI, CO_AQI, PM2.5_AQI, PM10_AQI) and creates a new column 'AQI_label' representing the maximum AQI among these pollutants. This label serves as the dependent variable for regression analysis.

```
from pyspark.sql import SparkSession, functions as F
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql import Row

spark = SparkSession.builder.appName("AQI_Regression_Fast").getOrCreate()
spark.sparkContext.setLogLevel("WARN")

df = (spark.read.option("header", True).option("inferSchema", True).csv("/content/part-*.csv"))

aqi_cols = [c for c in ["NO2_AQI","O3_AQI","SO2_AQI","CO_AQI","PM2_5_AQI","PM10_AQI"] if c in df.columns]
df = df.withColumn("AQI_label", F.array_max(F.array(*[F.col(c).cast("double") for c in aqi_cols]))).dropna(subset=["A

feature_cols = [c for c in ["NO2_Mean","O3_Mean","SO2_Mean","CO_Mean","PM2_5_Mean","PM10_Mean"] if c in df.columns]
df = df.dropna(subset=feature_cols).select(*(feature_cols + ["AQI_label"])).sample(False, 0.3, 42).cache()

train, test = df.randomSplit([0.8, 0.2], seed=42)

assembler = VectorAssembler(inputCols=feature_cols, outputCol="features_vec")
scaler = StandardScaler(inputCol="features_vec", outputCol="features_z", withMean=True, withStd=True)
lr = LinearRegression(featuresCol="features_z", labelCol="AQI_label", predictionCol="AQI_pred", maxIter=50, regParam=

model = Pipeline(stages=[assembler, scaler, lr]).fit(train)

pred = model.transform(test).cache()
rmse = RegressionEvaluator(labelCol="AQI_label", predictionCol="AQI_pred", metricName="rmse").evaluate(pred)
mae  = RegressionEvaluator(labelCol="AQI_label", predictionCol="AQI_pred", metricName="mae").evaluate(pred)
r2   = RegressionEvaluator(labelCol="AQI_label", predictionCol="AQI_pred", metricName="r2").evaluate(pred)
print(f"RMSE: {rmse:.4f}  MAE: {mae:.4f}  R2: {r2:.4f}")

coef_stage = model.stages[-1]
coef_df = spark.createDataFrame([Row(feature=f, coefficient=float(c)) for f, c in zip(feature_cols, coef_stage.coeffi
```

Figure 3.4: Feature assembly, scaling, and model training.

```
RMSE: 11.6549  MAE: 7.5212  R2: 0.6255
+---------+--------+--------+--------+---------+-------------------+
|NO2_Mean |O3_Mean |SO2_Mean|CO_Mean |AQI_label|AQI_pred           |
+---------+--------+--------+--------+---------+-------------------+
|-0.279167|0.03125 |0.0125  |0.137417|30.0     |30.30440163241441  |
|-0.154167|0.010125|0.021739|0.1     |18.0     |2.206747638298644  |
|0.0      |0.007542|0.0     |0.1     |12.0     |-1.0631346526626828|
|0.0      |0.012   |0.0     |0.066667|15.0     |4.741218782912227  |
|0.0      |0.012167|0.125   |0.404167|14.0     |6.312568235434199  |
|0.0      |0.014   |0.0     |0.05    |16.0     |7.338728969426327  |
|0.0      |0.0145  |0.55    |0.008333|19.0     |8.140627885372538  |
|0.0      |0.015375|0.8     |0.2     |16.0     |10.166945865663262 |
|0.0      |0.016083|0.033333|0.1     |18.0     |10.31780229036676  |
|0.0      |0.016125|0.0     |0.120833|19.0     |10.434944697287406 |
+---------+--------+--------+--------+---------+-------------------+
only showing top 10 rows
```

Figure 3.5: Predicted vs. actual AQI values (sample output).

3.4 Fredrick Damptey – Chi-Square Test for Feature Selection

I conducted a Chi-Square test of independence to evaluate the relationship between pollutant concentration levels and the frequency of unhealthy air days. Using PySpark's ChiSquareTest module, the analysis tested whether categorical pollutant categories (low, moderate, high) had a statistically significant association with unhealthy air conditions. To start, I set up and verified the Spark master and worker nodes across his virtual machines (VMs: 192.168.13.116 and 192.168.13.117), ensuring that distributed computation was operational for scalable data analysis.

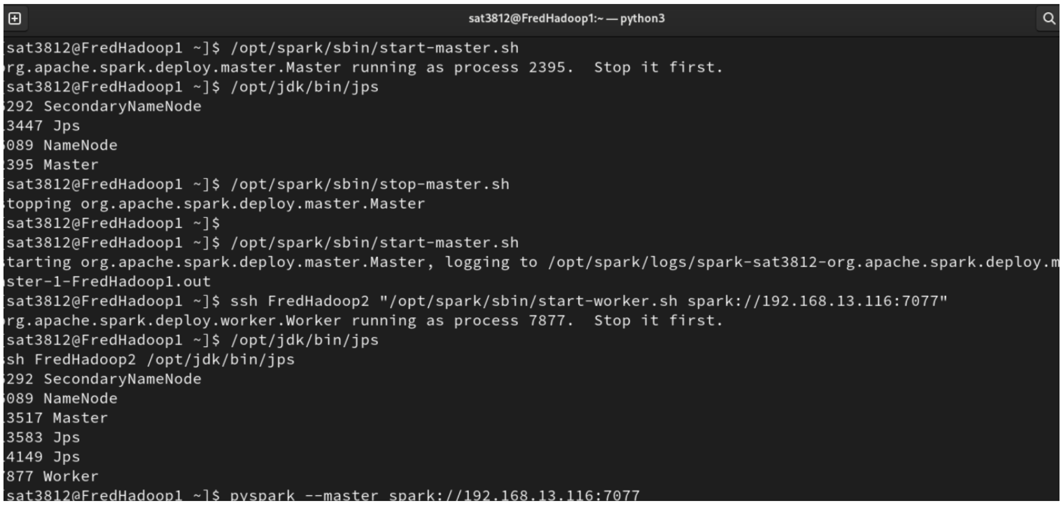


Figure 4.1: Fredrick's Spark Master Node (192.168.13.116) started successfully

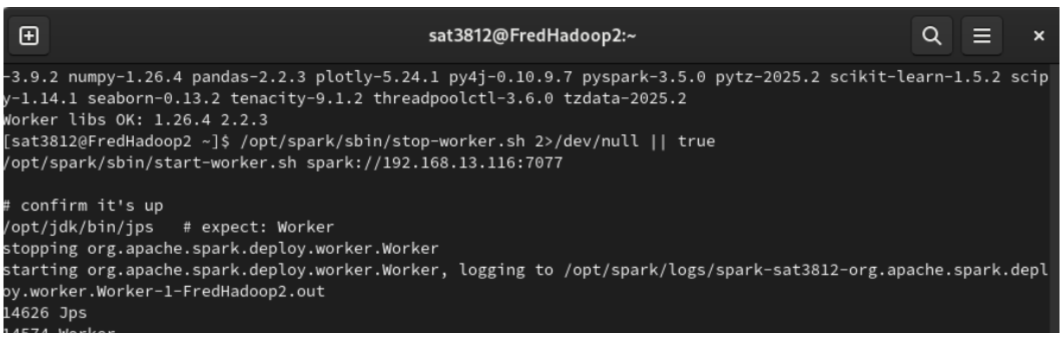


Figure 4.2: Fredrick's Worker Node (192.168.13.117) connected to Spark Master.

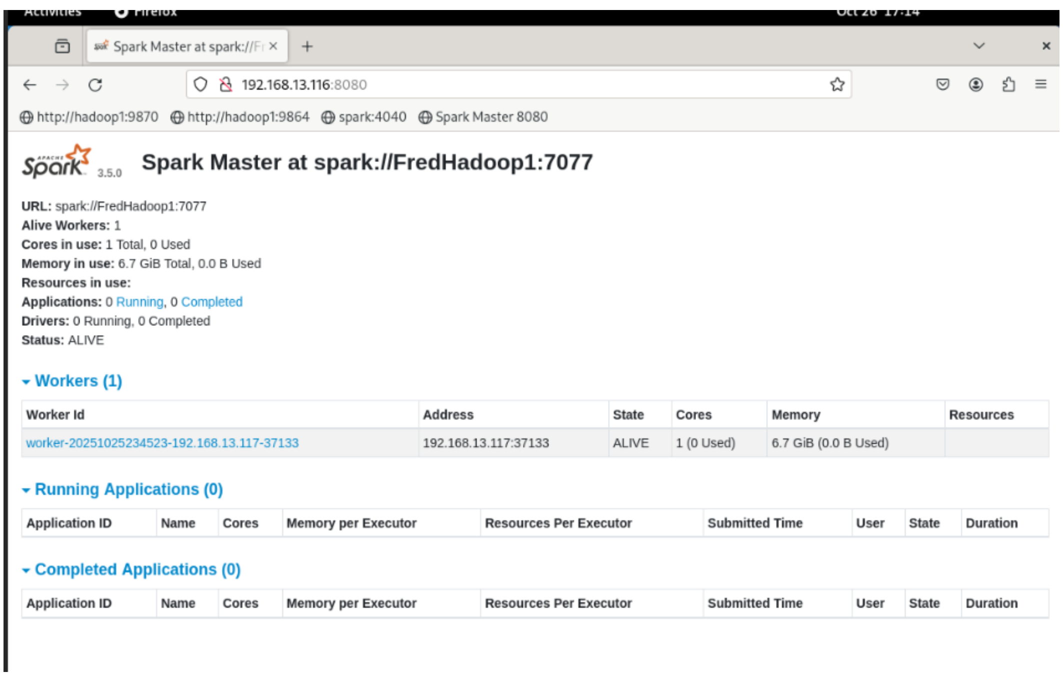


Figure 4.3: Spark Master Web UI (Fredrick) confirming one active worker node

```
>> for colname, (b1, b2) in biming.items():
..     cat_col = f"{colname}_cat"
..     cur = cur.withColumn(
..             cat_col,
..             F.when(F.col(colname) <= b1, "low")
..             .when(F.col(colname) <= b2, "moderate")
..             .otherwise("high")
..     )
..     idx_col = f"{cat_col}_idx"
..     idx = StringIndexer(inputCol=cat_col, outputCol=idx_col, handleInvalid="
kip").fit(cur)
..     tmp = idx.transform(cur).select("unhealthy", idx_col).na.drop()
..     feat = VectorAssembler(inputCols=[idx_col], outputCol="features").transf
rm(tmp)
..     ch = ChiSquareTest.test(feat, "features", "unhealthy").head()
..     results.append(Colname, float(ch.statistics[0]), int(ch.degreesOfFreedom
0]), float(ch.pValues[0])))
 File "<stdin>", line 14
   results.append(Colname, float(ch.statistics[0]), int(ch.degreesOfFreedom[0])
 float(ch.pValues[0])))
```

Figure 4.4: Air Quality Category Classification and Chi-Square Calculation

```
>>> selected_features = [chi_bins[i] for i in selected_idx]
>>> chi_df_sel = chi_df.where(col("feature").isin(selected_features)).orderBy("p_value")
>>> chi_df_sel.show(truncate=False)
+--------------------------+-------+------------------+
|feature                   |p_value|statistic         |
+--------------------------+-------+------------------+
|State_Code__imp__bin      |0.0    |2931.4101759446417|
|O3_1st_Max_Hour__imp__bin |0.0    |11338.239451518097|
|County_Code__imp__bin     |0.0    |8868.904900963105 |
|SO2_Mean__imp__bin        |0.0    |8652.410520138874 |
|Site_Num__imp__bin        |0.0    |3228.23554529234  |
|SO2_1st_Max_Value__imp__bin|0.0   |8607.87594267527  |
|NO2_Mean__imp__bin        |0.0    |10009.434742879457|
|SO2_1st_Max_Hour__imp__bin |0.0   |1598.2814840163253|
|NO2_1st_Max_Value__imp__bin|0.0   |11905.007217635484|
|SO2_AQI__imp__bin         |0.0    |2438.658628830249 |
|NO2_1st_Max_Hour__imp__bin |0.0   |5135.851901902335 |
|CO_Mean__imp__bin         |0.0    |2377.586885185572 |
|NO2_AQI__imp__bin         |0.0    |11749.014336383272|
|O3_Mean__imp__bin         |0.0    |127347.78869875579|
|O3_1st_Max_Value__imp__bin |0.0   |131153.98191949102|
+--------------------------+-------+------------------+
```

Figure 4.5: Chi-Square Test Results

3.5 Sucharitha reddy Dammareddygari – Dimensionality Reduction (PCA)

I used Principal Component Analysis (PCA) to reduce the dimensionality of the air quality dataset while preserving most of the important information. The dataset included many related pollutant measures, such as PM2.5, PM10, NO_2, SO_2, CO, and O_3. To start, I set up and verified the Spark master and worker nodes across his virtual machines (VMs: 192.168.13.116 and 192.168.13.117), ensuring that distributed computation was operational for scalable data analysis. Before applying PCA, all the data were standardized so that every feature had equal importance. Then, using Spark MLlib, the PCA model found the main components that explain most of the data variation. The results showed that the first few components captured most of the total information, which means we could work with fewer features without losing accuracy. This made our analysis faster, simpler, and easier to understand. Reduced data can also be used to train predictive models to classify air quality days as “healthy” or “unhealthy.”

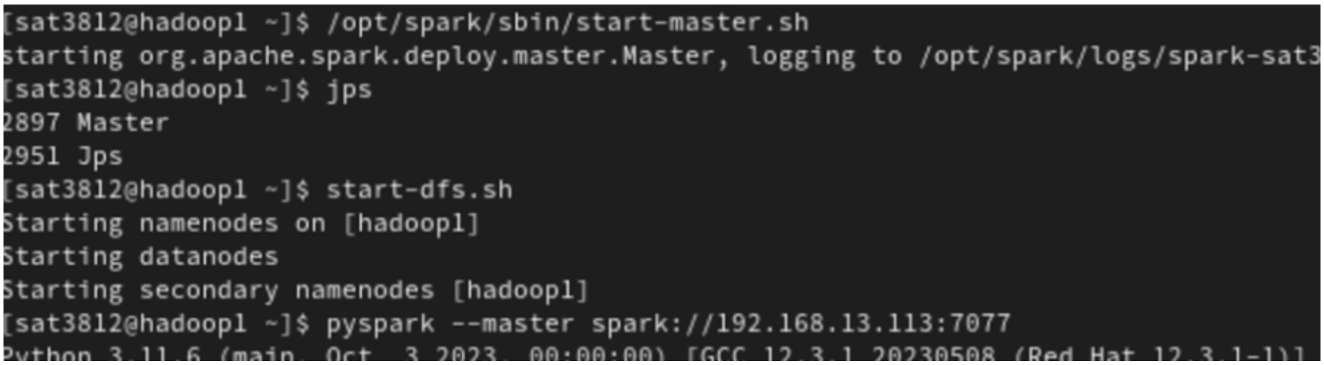


Figure 5.1: Sucharitha’s Spark Master Node (192.168.13.113) started successfully

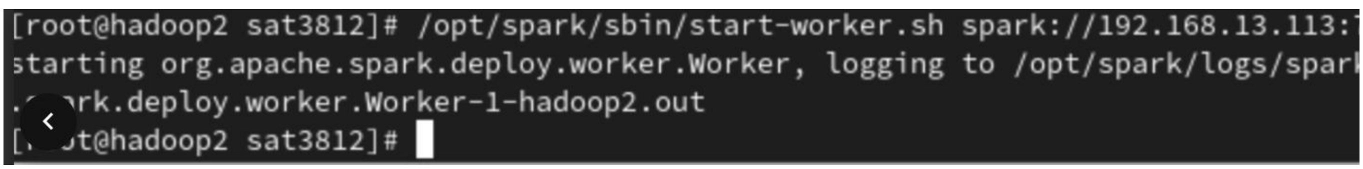


Figure 5.2: Sucharitha’s Worker Node (192.168.13.114) connected to Spark Master.

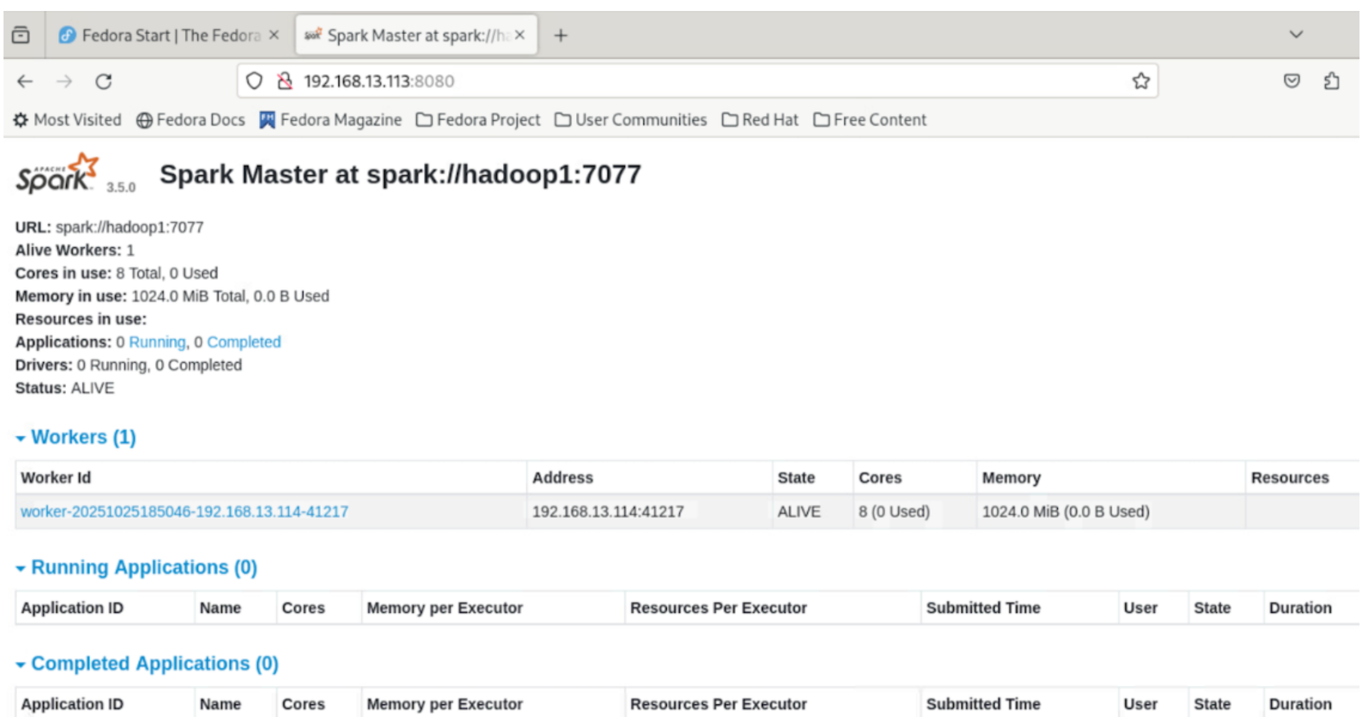


Figure 5.3: Spark Master Web UI (Sucharitha) confirming one active worker node

```
Explained variance ratio: [0.525783182686236, 0.22542532144494884, 0.16650776678532875]
+------------------+------------------+--------------------+--------+--------+--------+--------+
|pc_1              |pc_2              |pc_3                |NO2_Mean|O3_Mean |SO2_Mean|CO_Mean |
+------------------+------------------+--------------------+--------+--------+--------+--------+
|1.737732749912841 |0.4015872562323784|0.10359616296396407 |4.541667|0.038458|0.958333|0.025   |
|1.7451441500302567|0.4032987370179101 |0.09608461043974453 |4.541667|0.038458|0.958333|0.020833|
|1.7419062347178151|0.3912312407873393 |0.10764207430090147 |4.541667|0.038458|0.925   |0.025   |
|1.749317634835231 |0.3929427215728709 |0.10013052177668194 |4.541667|0.038458|0.925   |0.020833|
|0.6286862157919038|0.074522142095652l |-0.6099443236202006 |10.0    |0.025208|1.875   |0.095833|
|0.6286862157919038|0.0745221420956521 |-0.6099443236202006 |10.0    |0.025208|1.875   |0.095833|
|0.6286862157919038|0.0745221420956521 |-0.6099443236202006 |10.0    |0.025208|1.875   |0.095833|
|0.6286862157919038|0.0745221420956521 |-0.6099443236202006 |10.0    |0.025208|1.875   |0.095833|
|0.9764729676238928|1.496027585465394  |0.005548646957150516|8.583333|0.039958|4.375   |0.1     |
|0.9690615675064773|1.4943161046798625 |0.013060199481369883|8.583333|0.039958|4.375   |0.104167|
+------------------+------------------+--------------------+--------+--------+--------+--------+
only showing top 10 rows
```

Figure 5.4: Dimensionality Reduction

5. Conclusion

This project showed how PySpark can be used to handle and analyze very large air quality datasets quickly and efficiently. Using Spark on multiple virtual machines helped the team process more than two million pollution records faster than traditional tools.

Each team member worked on a specific part that built the project step by step. Mary cleaned the data to make sure it was accurate and complete. Dennis found how pollutants like NO_2, SO_2, CO, and O_3 are related to air quality. Uttam built regression models to predict the Air Quality Index from pollutant levels. Fredrick used the Chi-Square test to find which pollutants have the strongest effect on unhealthy air days. Sucharitha reduced the dataset size using PCA to make it easier and faster to analyze while keeping most of the important information.

The final results showed that NO_2 and PM2.5 were the main pollutants causing poor air quality. The project proved that Spark is very powerful for large scale environmental analysis and can be used for future air quality monitoring, prediction, and decision-making to protect public health.