Analysis of data from an environmental sensor network using Hadoop/Spark

MSc in CSTE, CIDA option Machine learning & Big Data

Cranfield University

Submitted by:

Nnamdi Daniel Aghanya - SID: 460020

Paul Fontanges - SID: 461720

Bobin Mathis - SID: 462056

Word count: 2786

1 Introduction

Globally, the rapid growth of urban areas is an ongoing phenomenon (Ahmad et al., 2021). Cities and Urban areas around the world are impacted by environmental air pollution, a serious public health concern (Zhang et al., 2022). The United Nations health agency notes that over 80% of people living in cities are exposed to air quality levels that are higher than those recommended by the WHO (UN, 2016). Harlan and Ruddell (2011), and Zhang et al. (2022) support this assessment by documenting that rapid urbanisation and industrial growth are associated with declining air quality. While governmental agencies provide essential air quality data, their efforts are often limited by cost constraints and geographic coverage, particularly in delivering real-time data gathering and monitoring capabilities (Zhang, Zhou, & Luo, 2021). To address these limitations, Environmental Sensor Networks (ESNs) have emerged as a pivotal tool for observing and managing various ecological and climatic parameters because of its low-cost and quick deployment features (Hart & Martinez, 2006).

ESNs consist of spatially distributed sensors that collect real-time data on factors such as temperature, crop growth, soil moisture, air quality, light intensity and so on through a network of interconnected sensor nodes (Akyildiz et al., 2002). The captured data is transmitted to the base station for data processing, resulting in the extraction of valuable data for many stakeholders.

ESNs produce large volume of spatiotemporal data due to the widespread deployment of sensor nodes (Dereszynski & Dietterich, 2011), requiring sophisticated distributed computing frameworks for processing and analysis (Akyildiz et al., 2002; Zhang, Zhou & Luo, 2021). Apache Spark is a robust and scalable open-source paradigm that employs a distributed computing strategy designed to quickly and efficiently process and analyse big data and extract

valuable information for stakeholders (Huang, Zhang and Zhai, 2022). It is economical, flexible, extensible, and stable.

Spark performs better than previous big data tools like Apache Hadoop and Google's MapReduce because of its optimized query execution and in-memory computing capabilities (Hagar and Gawali, 2022). One of the benefits of Apache Sparks is that it uses unified application programming interfaces (API), making it easier to develop applications. Secondly, the combination of processing tasks is better when using Spark (Zaharia et al., 2016). However, developers may encounter unusual difficulties if they wish to include logging statements to monitor big data applications because of the abstraction that Spark offers. In addition, Spark optimises the data processing pipeline by implementing lazy evaluation. Application performance may be adversely affected if the data processing pipeline is broken to capture intermediate information at each stage (Wang et al., 2021). Conclusively, Spark's overall performance is thought to be superior to Hadoop's (Samadi et al.,2016).

Using Apache Spark framework to process and analyse vast amount of air quality data from ESNs will help national regulators, public health and other organisations to accurately track and assess the current situation, appraise the effectiveness of improving air quality in cities, and take evidence-based actions to lessen or better manage the problem of air pollution, thereby combating climate change and advancing sustainable development initiatives (SDG 11, Sustainable Cities and Communities) of the United Nations (UNDP, 2024). From the foregoing, Apache Spark framework can be used to process and analyse large-scale air quality data stream, providing useful information and insights to interested parties.

1.1 Aim and objectives

In this assignment, I focus on using Apache Spark to analyse air quality data captured from a network of small environmental sensors, since it provides a holistic method capable of

embracing both spatial and temporal aspects, making it more efficient with the challenges of computational complexity (Zhang et al., 2021). To sum up, the coursework focuses on the following three important tasks:

- ✓ To identify top 10 countries in terms of current average air quality improvement over the previous 24 hours and each average air quality index.
- ✓ To group the data into smaller regions using an appropriate clustering algorithm (k-means). Then, the top 50 regions will be determined regarding air quality improvement over the previous 24 hours.
- ✓ To calculate the longest streaks of good air quality and display them as a histogram.

2 Methodology

In this section, I discuss the methodology used in implementing Spark.

2.1 Data Collection and Schema Definition

We collected environmental sensor data by readings from a worldwide network of small air quality sensors and processed through Apache Spark distributed computing framework (PySpark and Hadoop). The sensor measurements are sourced from the *Sensor.Community* platform (Sensor.Community | Li, 2024) that transmits data in *JSON* format through dedicated *API* endpoints for both current and historical measurements. It needs:

- Current (Last 5 min average): https://data.sensor.community/static/v2/data.json
- Historical (Last 24 hr average): https://data.sensor.community/static/v2/data.24h.json

The sensor data schema is defined to obtain required set of attributes which is essential for capturing the analysis. Each data record contains the followings:

- **Sensor Identifier (s_i):** A unique ID for each sensor.

- **Geographic Coordinates ($l_i = (\phi_i, \lambda_i)$):** Latitude ($\phi_i$) and longitude ($\lambda_i$) of the sensor's location.
- **Timestamp (t_i):** The date and time when the measurement was recorded.
- **Sensor Measurements (v_i):** Particulate matter measurements, specifically $PM_{2.5}$ and PM_{10} values, represented as "P2" and "P1" in the data, respectively.

In figure 1 below, we report a schema in Spark to mirror the JSON data structure, ensuring correct data types and enabling efficient processing. Data is parsed and loaded into *Spark DataFrames* for both current and historical datasets, allowing for scalable data extraction, manipulation, processing and analysis.

```
sensor_schema = StructType([
    StructField("value_type", StringType(), True),
    StructField("value", StringType(), True),
])

schema = StructType([
    StructField("id", LongType(), True),
    StructField("timestamp", StringType(), True),
    StructField("location", StructType([
        StructField("id", LongType(), True),
        StructField("latitude", StringType(), True),
        StructField("longitude", StringType(), True),
        StructField("country", StringType(), True),
    ])),
    StructField("sensordatavalues", ArrayType(sensor_schema), True),
])
```

Figure 1. PySpark schema code mapping the sensor data JSON structure to structured types.

2.2 Air Quality Index Calculation

The Air Quality Index (AQI) is based on both $PM_{2.5}$ and PM_{10} particulate matter measurements. For each measurement, we compute two individuals standardized AQI values according to the UK air quality index standards. The overall AQI is then selected by choosing

the maximum value between the two measurements (see Table 1). This method ensures that the AQI reflects the level of the most important pollutant.

Table 1. UK Air Quality Index (AQI) categories and corresponding PM2.5 and PM10 concentration ranges (adapted from Committee on the Medical Effects of Air Pollutants, 2011, as cited in Li, 2024)

AQI Level	Category	$PM_{2.5}$ Range ($\mu g/m^3$)	PM_{10} Range ($\mu g/m^3$)
1	Excellent	0-11	0-16
2	Very Good	12-23	17-33
3	Good	24-35	34-50
4	Moderate	36-41	51-58
5	Fair	42-47	59-66
6	Poor	48-53	67-75
7	Very Poor	54-58	76-83
8	Bad	59-64	84-91
9	Very Bad	65-70	92-100
10	Hazardous	>70	>100

The overall AQI for each measurement is determined by the following equation:

$$\mathrm{AQI} = \max\left(\mathrm{AQI}_{\mathrm{PM}_{2.5}}\left(v_{\mathrm{PM}_{2.5}}\right),\ \mathrm{AQI}_{\mathrm{PM}_{10}}\left(v_{\mathrm{PM}_{10}}\right)\right) \tag{1}$$

By selecting the greater of the two pollutants to represent the AQI, this method most accurately adheres to health and environmental safety standards.

2.3 Temporal Analysis Framework

To examine the diurnal pattern of air quality and identify improvements, we calculate the average AQI values for time windows specific to each country. The mean AQI for each country c and on date t is defined as follows:

$$\mathrm{AQI}_{\mathrm{daily}}(c,t) = \frac{1}{N(c,t)} \sum_{i=1}^{N(c,t)} \mathrm{AQI}_i \tag{2}$$

where:

- $N(c,t)$ is the number of AQI measurements for the country (c) on date (t).
- AQI_i is the AQI value for the i^{th} measurement on that date.

We quantified air quality improvement by comparing the average AQI of the current day with that of the previous day, which is given as:

$$\text{Improvement}(c,t) = \text{AQI}_{\text{daily}}(c,t-1) - \text{AQI}_{\text{daily}}(c,t) \quad (3)$$

A positive value means an improvement in air quality, and vice versa for negative values.

We conduct a temporal analysis to identify the top 10 countries that showed the highest air quality improvements in the last 24 hours, along with their current average AQI.

2.4 Data Preprocessing

After collecting the data, the next phase is data preprocessing, which entails verifying the information and eliminating erroneous and missing data. The aim is to clean the dataset that can be distributed in a cluster and processed by Spark.

To capture spatial variations across countries and find unique air quality regions as well as account for the Earth's curvature and the convergence of meridians towards the poles, we transform the longitude values by the cosine of the latitude as defined below:

$$\lambda_{\text{scaled}} = \lambda \times \cos(\phi) \quad (4)$$

This adjustment ensures that the Euclidean distance calculations in the clustering algorithm accurately reflect true geographical distances.

2.5 Spatial Clustering Analysis

To capture spatial variations across countries and find unique air quality regions, k-means clustering (k=100) was performed on sensor coordinates after rescaling longitudes. Clustering techniques, such as k-means integrated with time-series analysis of air quality measurements, can assist in identifying key data features found within the context of the environment (Lv et al., 2016; Zheng et al., 2015)., Thus, we use k=100 clusters to create a compromise granularity that retains local air quality signal features but does not overly segment the data (Meena, Bairwa & Agarwal, 2024). Though, processing these kinds of data from environmental sensors may bring some challenges due to the real-time streaming input nature while maintaining computational efficiency (Gama et al., 2014); however, Karau et al. (2015) point out that Spark addresses the issues through its RDD-based architecture and implicit parallelization capabilities (Zaharia et al., 2012).

In the analysis, the k-means clustering minimizes the sum of squared distances between each sensor's location and the centroid of its assigned cluster to which it is a member. The mathematical expression is shown in Eq. (5):

$$\text{minimize} \sum_{j=1}^{k} \sum_{i \in C_j} \left(\left(\phi_i - \mu_{\phi_j} \right)^2 + \left(\lambda_{\text{scaled}_i} - \mu_{\lambda_{\text{scaled}_j}} \right)^2 \right) \tag{5}$$

Where:

- C_j is the set of sensors assigned to cluster j.
- $(\mu_{\phi_j}, \mu_{\lambda_{\text{scaled}_j}})$ is the centroid of cluster j.

2.6 Regional AQI Analysis

After clustering, we calculated the average AQI for each cluster (region) and identified improvements over the previous 24 hours using the same temporal framework for countries

(see Equation (2) & (3)). This analysis enables us to determine the top 50 regions in terms of air quality improvement.

2.7 Streak Analysis

To detect stretches of sustained good air quality, a streak analysis was performed on the AQI time series data from each sensor location. A "good" AQI is defined as an index value between 1 and 3, indicating low pollution levels.

2.7.1 Streak Counting

We calculated the number of consecutive days with an AQI in the good range for each sensor location as follows:

Condition	Value
$\text{AQI}_i \leq 3$	$S_{i-1} + 1$
$\text{AQI}_i > 3$	0

Table 1. A streak of good air quality is defined when $\text{AQI}_i \leq 3$, corresponding to the 'Low' category of air quality (Li, 2024).

where S_i is the streak length at time t_i, and AQI_i is the AQI value at time t_i.

2.7.2 Histogram Visualization

Next, we combine the streak lengths across all sensor locations and plotted their distribution (using Matplotlib and Seaborn), as a histogram H with 20 bins between zero and the maximum streak length found, as defined:

$$H_b = |\{s \in S \mid \text{bin}_b(s)\}|$$

Where H_b is the count of streaks falling into bin b, and $\text{bin}_b(s)$ assigns streak length s to the appropriate histogram bin. This visualization provided insights into the frequency and duration of good air quality periods over the data collection period.

GeoPandas was used to plot sensor locations on a world map, visualizing regional AQI distributions. Results were saved as CSV files, and visualizations were exported as PNG images with timestamps for versioning.

2.5.3 Data Collection Period

The streak analysis was performed on the time period covered by the historical dataset (*data.24h.json*): representing the last 24 hours measurements. There was enough data to find and analyze stretches of clean air during this time period. The time-frame for the data collected from *sensor.community* is between 2024:11:22 02:09:13 to 2024:11:24 02:22:11.

3. Results and Discussion

In this section, we will discuss the results of our analysis.

3.1 Dataset description

The initial stage of the analysis involves cleaning the raw data (preprocessing task), as explained in section 2 above. Figure 2 below shows air quality data used for the analysis. From the figure, after eliminating erroneous and missing data, about 60.4% or 8,010,884 records (out of 13,255,268 data collected) of average last 24 hours data remained.

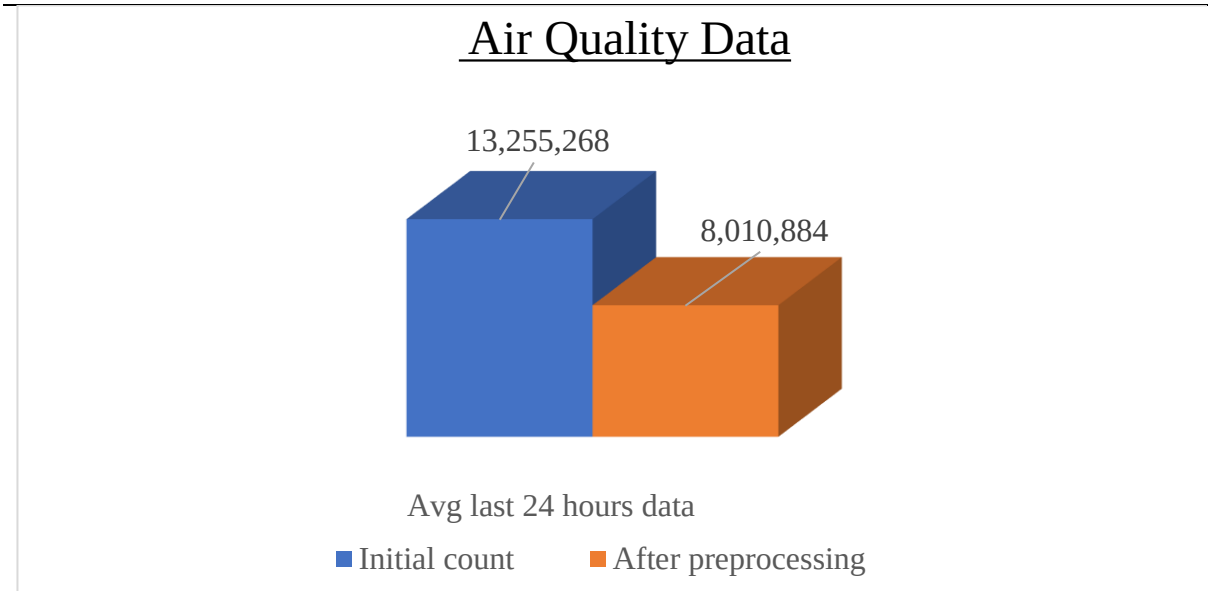

Figure 2. Air quality data used for the analysis

3.1 Air Quality Improvement by Country

We analyzed the air quality data to identify the top countries with the most significant improvements in their Air Quality Index (AQI) over the previous 24 hours. The AQI was computed from both $PM_{2.5}$ and PM_{10} measurements to provide a holistic understanding of particulate pollution.

Table 3 presents the top 10 countries by the amount of AQI improvement (at the time of this analysis). Top of the list with the greatest air quality improvement is Uzbekistan (UZ), which showed the largest average change in AQI, from 7.231 the previous day to 3.024 in the current day, recording and an AQI improvement of 4.207. This is closely followed by Liberia (LR) with an improvement of 2.550. The bottom three on the table of the top 10 countries greatest air quality improvement are Bosnia (BA), Cyprus (CY) and Cambodia (KH), recording and an AQI improvement of 0.880, 0.840 and 0.823, respectively.

Country Code	Date	Num Sensors	Previous Day AQI	Avg AQI	AQI Improvement
UZ	2024-11-24	286	7.008	2.405	4.603
LR	2024-11-24	99	9.575	5.916	3.658
ID	2024-11-24	305	5.268	2.491	2.776
KG	2024-11-24	573	6.996	5.207	1.788
HR	2024-11-24	898	2.900	1.369	1.531
KZ	2024-11-24	658	5.352	3.875	1.477
RS	2024-11-24	12129	4.934	3.468	1.466
BA	2024-11-24	1922	5.278	3.960	1.319
KH	2024-11-24	100	2.190	1.16	1.030
CY	2024-11-24	396	2.074	1.053	1.02

The AQI improvements in these countries could be attributable to factors such as favorable meteorological conditions, effective environmental policies implemented, or reduction in industrial activities during the period of data collection. Notably, UZ, LR and KG had previous day AQI values at greater than 7 (that is, very poor to bad), indicating high pollution levels, and their substantial improvements represent a positive shift towards better air quality.

3.2 Air Quality Improvement by Region

To further aggregate data for localized evaluation of air quality improvements, we clustered the data into smaller regions using geo-coordinates derived from the sensor data.

Using the elbow test to determine 60 clusters effectively segments the sensor data into 60 geographically distinct regions, ensuring each cluster captures meaningful spatial patterns in air quality measurements. This optimal number of clusters balances the complexity of the model with the need for detailed regional analysis, allowing for precise identification of areas experiencing significant air quality improvements.

Consequently, we performed K-means clustering (k = 60) taking into account the Earth's curvature by scaling longitude values. Table 4 shows the lists of top 50 regions in terms of air quality improvement. From the table, Region 39 demonstrates the most significant improvement, with an AQI decrease of 2.621. This region, centered at latitude 52.021 and longitude 108.867, corresponds to an area in Easten Europe (Russia), aligning with the country-level attempt been made to improve environmental safety (IQAir, 2024).

Cluster ID	Num of Sensors	Center Lat.	Center Long.	Latitude	Longitude	Prev Day AQI	Cur. Day AQI	AQI Improvement
39	1213	52.021	108.867	52.020	108.862	4.207	1.586	2.621
14	366	-5.178	106.080	-5.195	106.048	5.123	2.847	2.276
34	1855	40.899	73.852	40.903	73.683	6.552	5.182	1.371
15	44139	46.660	19.501	46.665	19.501	2.645	2.106	0.538
38	30635	50.001	19.671	50.000	19.667	1.944	1.448	0.496
22	1196	32.167	35.302	32.149	35.386	1.726	1.293	0.434
50	27332	52.094	21.012	52.092	21.010	1.829	1.442	0.387
21	603	41.568	129.761	41.581	128.741	2.863	2.498	0.364
18	1784	0.415	-0.701	0.410	-0.689	1.697	1.341	0.356
23	5132	40.043	16.440	40.042	16.405	1.579	1.242	0.337
19	22596	51.051	17.001	51.051	17.003	1.602	1.274	0.328
65	14132	45.913	12.605	45.914	12.607	1.758	1.460	0.297
45	21137	53.302	18.034	53.303	18.021	1.769	1.471	0.297
17	413	-19.060	30.470	-19.141	30.048	1.524	1.228	0.296
59	2681	41.951	-82.216	41.947	-81.991	1.693	1.438	0.255
47	629	30.840	-17.850	30.943	-17.838	1.721	1.512	0.209
7	4468	50.217	30.961	50.216	30.972	1.670	1.465	0.205
54	9384	54.541	-3.988	54.553	-4.026	1.666	1.480	0.186
52	23055	45.001	10.204	44.999	10.204	1.408	1.235	0.173
60	11381	43.597	0.876	43.591	0.875	1.535	1.371	0.164
40	8786	57.360	12.100	57.370	12.110	1.436	1.286	0.150
25	9062	48.113	-2.361	48.112	-2.358	1.738	1.596	0.142
62	5085	42.510	12.571	42.508	12.566	1.383	1.281	0.102
53	23736	55.886	38.298	55.881	38.279	1.180	1.082	0.098
29	32066	47.867	15.884	47.866	15.880	1.367	1.280	0.088
68	1972	37.846	-7.570	37.854	-7.566	1.155	1.092	0.064
31	4042	60.182	28.407	60.178	28.429	1.075	1.013	0.062
9	2663	42.336	-4.660	42.331	-4.626	1.559	1.503	0.056
51	55228	48.219	11.703	48.219	11.702	1.261	1.212	0.049
56	36216	50.854	13.491	50.860	13.473	1.230	1.182	0.048
46	1561	46.564	-122.825	46.560	-122.785	1.148	1.107	0.041
32	593	12.173	-67.914	12.211	-67.637	1.036	1.005	0.031
61	11444	48.920	2.302	48.921	2.299	1.044	1.014	0.030
24	255033	52.009	5.324	52.007	5.312	1.251	1.224	0.027
11	105780	42.579	23.793	42.578	23.785	1.516	1.490	0.026
4	116220	48.492	8.975	48.492	8.974	1.311	1.285	0.025
27	13837	45.659	6.041	45.657	6.032	1.320	1.301	0.019
30	66376	49.860	8.328	49.858	8.327	1.201	1.185	0.016
41	32516	49.767	10.913	49.761	10.911	1.227	1.211	0.016
13	108736	51.181	6.741	51.182	6.740	1.327	1.315	0.011
1	603	-35.385	149.295	-35.380	149.288	1.007	1.002	0.005
69	2567	64.705	19.229	64.688	19.011	1.004	1.000	0.004
55	3838	44.749	38.041	44.752	37.996	1.004	1.002	0.002
43	124	-88.493	-24.609	-88.493	-24.609	1.000	1.000	0.000
49	126	64.130	-21.889	64.130	-21.889	1.000	1.000	0.000
67	380	-27.795	158.281	-27.771	156.172	1.000	1.000	0.000
20	43737	51.980	9.501	51.979	9.504	1.265	1.266	-0.001
26	2260	44.522	-72.778	44.537	-72.724	1.027	1.031	-0.003
48	3856	55.749	25.636	55.756	25.646	1.048	1.054	-0.006
63	10101	59.381	17.529	59.383	17.530	1.154	1.165	-0.010

Most regions from Table 4 with significant improvements are predominantly located in Europe and Asia. The spatial distribution of these improvements suggests that regional factors such as local policies, emission reductions, or weather patterns may have contributed to better air quality.

3.3 Longest Streaks of Good Air Quality

Figure 3 illustrates the distribution of the longest streaks across all sensor locations. with most streaks clustering between 100-200 hours and showing a right-skewed distribution where a few locations maintain good air quality for up to 600 hours. The mean and median streak length are 134.4 and 140, respectively; with the histogram reveals that there are considerable variations in the streak length and a notable peak around the 150-hour mark.

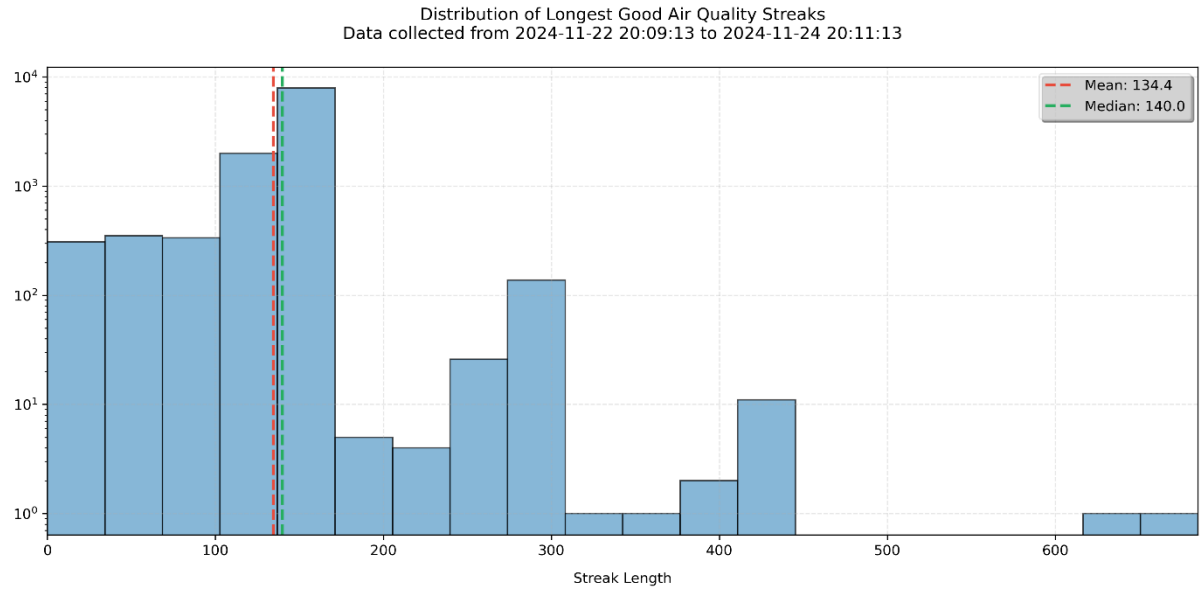

Figure 3. Distribution of Longest Streaks of Good Air Quality (AQI 1-3).

3.4 Geographical Visualization of AQI Data Points

To visualize the spatial distribution of the AQI data points, we plotted the sensor locations on a world map. Figure 4 below reveals that the sensors are widely distributed across Europe and parts of Asia, with clusters corresponding to densely populated areas. This visualization helps

in understanding the spatial coverage of the sensor network and identifying regions with lower and higher sensor densities.

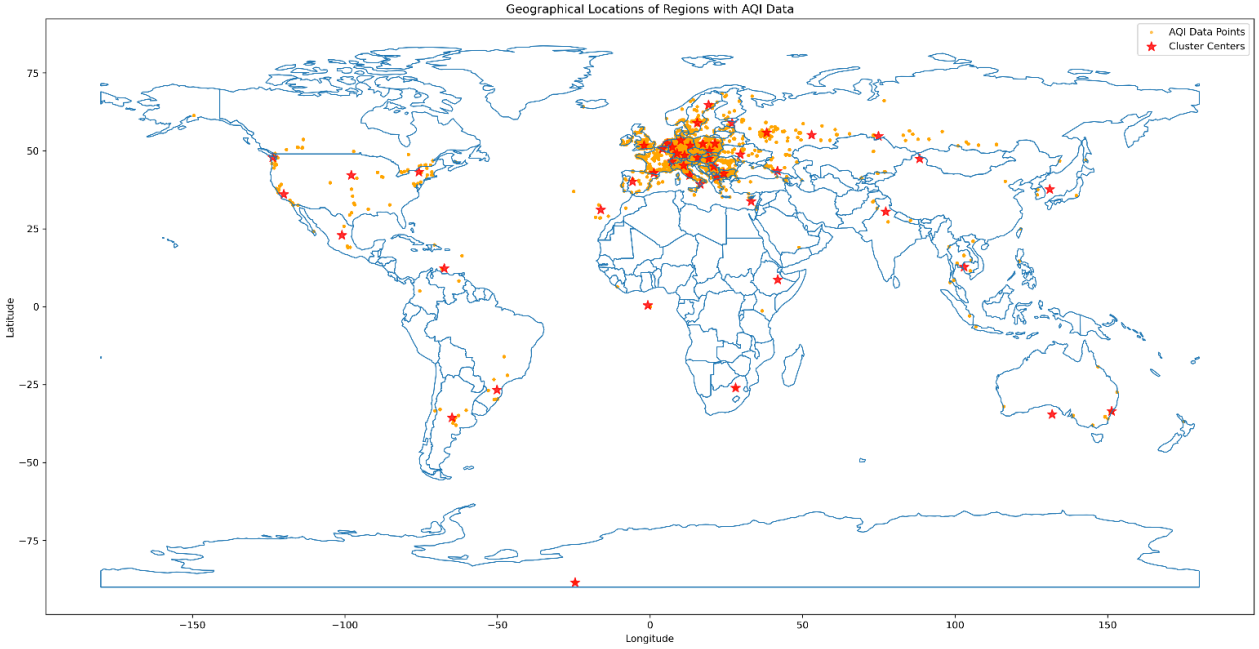

Figure 4. Geographical Locations of Regions with AQI Data Points.

3.5 Discussion

The significant air quality improvements observed in certain countries and regions may be influenced by various factors, including:

- Meteorological Conditions: Changes in weather patterns, such as increased rainfall or wind, can disperse pollutants and improve air quality.
- Policy Measures: Implementation of emission reduction policies, traffic restrictions, or industrial shutdowns can lead to rapid improvements.

The regional analysis emphasizes the importance of localized strategies for air quality management. Clustering sensors into regions allows for targeted interventions and monitoring of specific areas that may require additional attention.

The streak analysis indicates that while some locations experience consecutive periods of good air quality, these are generally short-lived. This underscores the need for continuous efforts to maintain and improve air quality, as well as the importance of real-time monitoring to promptly address pollution spikes.

Overall, the results highlight the dynamic nature of air quality and the effectiveness of combining temporal and spatial analyses to gain comprehensive insights. Further research could involve correlating these findings with emission inventories, traffic data, and meteorological conditions to better understand the underlying causes of the observed improvements.

Conclusion

The aim of this assignment was to analyse global air quality data using Apache Spark and to identify significant patterns and improvements in the Air Quality Index (AQI). Our analysis yielded the following key findings:

The analysis shows that top Countries that exhibited the most substantial improvements in air quality for the period under review include Uzbekistan (UZ) and Liberia (LR). The bottom three on the table of the top 10 countries greatest air quality improvement are Bosnia (BA), Cyprus (CY) and Cambodia (KH). Also, clustering of sensor data reveals that regions centred around Central Asia and Europe showed notable AQI improvements.

References

Ahmad, M., Jiang, P., Murshed, M., Shehzad, K., Akram, R., Cui, L., & Khan, Z. (2021). Modelling the dynamic linkages between eco-innovation, urbanization, economic growth and ecological footprints for G7 countries: does financial globalization matter?. *Sustainable Cities and Society*, *70*, 102881.

Akyildiz, I.F., Su, W., Sankarasubramaniam, Y. and Cayirci, E. (2002) ' Wireless sensor networks: a survey', *Computer Networks*, 38(4), pp. 393-422. https://doi.org/10.1016/S1389-1286(01)00302-4

Dereszynski, E.W. and Dietterich, T.G. (2011) 'Spatiotemporal Models for Data-Anomaly Detection in Dynamic Environmental Monitoring Campaigns', *ACM Transactions on Sensor Networks*, 8(1), pp. 1-36. doi: 10.1145/1993042.1993045

Gama, J., Indrė Žliobaitė, Bifet, A., Mykola Pechenizkiy and Abdelhamid Bouchachia (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, [online] 46(4), pp.1–37. doi: https://doi.org/10.1145/2523813.

Hagar, A. A., & Gawali, B. W. (2022). Apache spark and deep learning models for high-performance network intrusion detection using CSE-CIC-IDS2018. *Computational Intelligence and Neuroscience*, *2022*(1), 3131153. https://doi. org/10.1155/2022/3131153.

Harlan, S. L., & Ruddell, D. M. (2011). Climate change and health in cities: impacts of heat and air pollution and potential co-benefits from mitigation and adaptation. *Current opinion in environmental sustainability*, *3*(3), 126-134. https://doi.org/10.1016/j.cosust.2011.01.001

Hart, J.K. and Martinez, K. (2006) 'Environmental Sensor Networks: A revolution in the earth system science?', *Earth-Science Reviews*, 78(3-4), pp. 177-191. doi: 10.1016/j.earscirev.2006.05.001

Huang, X., Zhang, H., & Zhai, X. (2022). A novel reinforcement learning approach for spark configuration parameter optimization. *Sensors*, *22*(15), 5930. https://doi.org/10.3390/s22155930

IQAir (2024). Air quality in Russia: Air quality index (AQI) and PM2.5 air pollution in Russia. Available at: https://www.iqair.com/russia (Accessed 24/11/2024)

Karau, H., Konwinski, A., Patrick, W. and Zaharia, M. (2015). *Spark LIGHTNING-FAST DATA ANALYSIS*. [online] Available at: https://cs.famaf.unc.edu.ar/~damian/tmp/bib/Learning_Spark_Lightning-Fast_Big_Data_Analysis.pdf.

Li J. Analysis of data from an environmental sensor network using Hadoop/Spark [Assignment brief]. Cranfield University, MSc in CSTE CIDA option Machine Learning & Big Data. Canvas VLE: canvas.cranfield.ac.uk; 2024 [cited 2024 Nov 9]. Available from: https://canvas.cranfield.ac.uk

Lv, Z., Hu, Y., Zhong, H., Wu, J., Li, B. and Zhao, H. (2010) 'Parallel K-Means clustering of remote sensing images based on MapReduce,' in Lecture notes in computer science, pp. 162–170. https://doi.org/10.1007/978-3-642-16515-3_21.

Meena, K.K., Bairwa, D. and Agarwal, A. (2024). A machine learning approach for unraveling the influence of air quality awareness on travel behavior. *Decision Analytics Journal*, [online] 11, p.100459. doi: https://doi.org/10.1016/j.dajour.2024.100459.

Samadi, Y., Zbakh, M., & Tadonki, C. (2016). Comparative study between Hadoop and Spark based on Hibench benchmarks. In *2016 2nd International Conference on Cloud*

Computing Technologies and Applications (CloudTech) (pp. 267-275). IEEE. https://doi.org/10.1109/CloudTech.2016.7847709

Sensor.Community API Documentation on GitHub: https://github.com/opendata-stuttgart/meta/wiki/EN-Home, https://github.com/opendata-stuttgart/meta/wiki/EN-APIs

United Nations (UN) (2016). UN health agency warns of rise in urban air pollution. Available at: https://www.un.org/sustainabledevelopment/blog/2016/05/un-health-agency-warns-of-rise-in-urban-air-pollution-with-poorest-cities-most-at-risk/ (Accessed on 12/11/2024).

United Nations Development Programme (2024). Sustainable Development Goals. Available at: https://www.undp.org/sustainable-development-goals. (Accessed on 21/11/2024).

Wang, N., Chen, F., Yu, B., & Wang, L. (2022). A Strategy of Parallel SLIC Superpixels for Handling Large-Scale Images over Apache Spark. *Remote Sensing*, *14*(7), 1568. https://doi.org/10.3390/rs14071568

Wang, Z., Zhang, H., Chen, T. H., & Wang, S. (2021). Would you like a quick peek? providing logging support to monitor data processing in big data applications. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 516-526). https://doi.org/10.1145/3468264.3468613

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S. & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (pp. 15-28).

Zhang, X., Han, L., Wei, H., Tan, X., Zhou, W., Li, W. and Qian, Y. (2022) 'Linking urbanization and air quality together: A review and a perspective on the future sustainable urban development', *Journal of Cleaner Production*, 346, 130988. https://doi.org/10.1016/j.jclepro.2022.130988

Zhang, X., Zhou, Y. and Luo, J. (2021) 'Deep learning for processing and analysis of remote sensing big data: a technical review', *Big Earth Data*, 6(4), pp. 527–560. https://doi.org/10.1080/20964471.2021.1964879.

Zheng, Y., Yi, X., Li, M., Li, R., Shan, Z., Chang, E. and Li, T. (2015) 'Forecasting Fine-Grained Air Quality Based on Big Data', in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Sydney, NSW, Australia. New York: ACM, pp. 2267-2276. https://doi.org/10.1145/2783258.2788573

Appendix

Code

```
import pyspark
import geopandas as gpd
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
```

```
from pyspark.sql.types import *
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
import requests
from datetime import datetime, timedelta
import os
import time
import numpy as np
import matplotlib.pyplot as plt
from pyspark.sql.window import Window
from pyspark.sql.functions import udf
import threading
import json
from pyspark.sql.functions import col, lag, sum as Fsum, when, to_date, datediff


def fetch_data(url):
    """self explanatory by name
    @ param: None
    """
    try:
        response = requests.get(url)
        response.raise_for_status()
        print(f"Successfully fetched data from {url}")
        return response.json()
    except requests.RequestException as e:
        print(f"Error fetching data from {url}: {e}")
        return []


def filter_invalid_coordinates(df):
    """filter
    @ param:   df with latitude and longitude columns
    @ returns: df contains valid coordinates
    """
    return df.filter(
        (F.col("latitude").isNotNull())
        & (F.col("longitude").isNotNull())
        & (F.col("latitude") != 0.0)
        & (F.col("longitude") != 0.0)
    )


def collect_data_over_time(output_dir, duration_hours=0.00001, interval_minutes=0.001):
    """self explanatory by name
    @ param: How long you want to run it (duration), interval breaks
    """
    print(f"Starting data collection for {duration_hours} hours...")
    # ensure output directory exists
    os.makedirs(output_dir, exist_ok=True)
    # calculate end time
    start_time = datetime.now()
    end_time = start_time + timedelta(hours=duration_hours)
    while datetime.now() < end_time:
        current_time = datetime.now()
        # fetch current data
        print(f"\nFetching data at {current_time.strftime('%Y-%m-%d %H:%M:%S')}")
        try:
            current_response = fetch_data(
                "https://data.sensor.community/static/v2/data.json"
            )
            historical_response = fetch_data(
                "https://data.sensor.community/static/v2/data.24h.json"
            )
            if current_response and historical_response:
                # save current data
```

```
                current_filename = os.path.join(
                    output_dir,
                    f"current_data_{current_time.strftime('%Y%m%d_%H%M%S')}.json",
                )
                with open(current_filename, "w") as f:
                    json.dump(current_response, f)
                historical_filename = os.path.join(
                    output_dir,
                    f"historical_data_{current_time.strftime('%Y%m%d_%H%M%S')}.json",
                )
                with open(historical_filename, "w") as f:
                    json.dump(historical_response, f)
                print(f"Saved data to {current_filename} and {historical_filename}")
            else:
                print("No data fetched.")
        except Exception as e:
            print(f"Error during data collection: {str(e)}")
        next_collection = current_time + timedelta(minutes=interval_minutes)
        sleep_seconds = (next_collection - datetime.now()).total_seconds()
        if sleep_seconds > 0:
            print(f"Waiting {interval_minutes} minutes until next collection...")
            time.sleep(sleep_seconds)
    print("\nData collection completed!")


def process_data(df):
    """process raw sensor data into a structured format.
    @ param:   raw dataframe
    @ return:  processed dataframe"""
    data = df.select(
        "location.country",
        "location.latitude",
        "location.longitude",
        "timestamp",
        F.explode("sensordatavalues").alias("sensordata"),
    ).select(
        F.col("country"),
        F.col("latitude").cast(FloatType()),
        F.col("longitude").cast(FloatType()),
        F.col("timestamp"),
        F.to_date("timestamp").alias("date"),
        F.col("sensordata.value").cast(FloatType()).alias("value"),
        F.col("sensordata.value_type"),
    )
    print(f"After processing size: {data.count():,} records")
    return data


def pivot_data(data):
    """pivot the data to create separate columns for different measurement types.
    @ param:  data processed dataframe
    @ return: pivoted dataframe"""
    # filter only the necessary value_types
    data_filtered = data.filter(F.col("value_type").isin(["P1", "P2"]))
    data_pivot = (
        data_filtered.groupBy("country", "latitude", "longitude", "timestamp", "date")
        .pivot("value_type")
        .agg(F.first("value"))
    )
    print(f"Post-pivot DataFrame size: {data_pivot.count():,} records")
    return data_pivot


def calculate_aqi(pm25, pm10):
    """calculate air quality index
    @ param:  pm25: PM2.5 concentration value
```

```
            pm10: PM10  concentration value
    @ return: AQI value from 1-10"""
    pm25_ranges = [
        (0, 11, 1),
        (12, 23, 2),
        (24, 35, 3),
        (36, 41, 4),
        (42, 47, 5),
        (48, 53, 6),
        (54, 58, 7),
        (59, 64, 8),
        (65, 70, 9),
        (70, float("inf"), 10),
    ]
    pm10_ranges = [
        (0, 16, 1),
        (17, 33, 2),
        (34, 50, 3),
        (51, 58, 4),
        (59, 66, 5),
        (67, 75, 6),
        (76, 83, 7),
        (84, 91, 8),
        (92, 100, 9),
        (100, float("inf"), 10),
    ]
    aqi_pm25 = None
    if pm25 is not None:
        for low, high, index in pm25_ranges:
            if low <= pm25 <= high:
                aqi_pm25 = index
                break
    aqi_pm10 = None
    if pm10 is not None:
        for low, high, index in pm10_ranges:
            if low <= pm10 <= high:
                aqi_pm10 = index
                break
    if aqi_pm25 is not None and aqi_pm10 is not None:
        return max(aqi_pm25, aqi_pm10)
    elif aqi_pm25 is not None:
        return aqi_pm25
    elif aqi_pm10 is not None:
        return aqi_pm10
    else:
        return None


aqi_udf = udf(calculate_aqi, IntegerType())


def save_to_csv(df, filepath, timestamp=None):
    """save df to csv
    @ param: file path
    """
    try:
        if timestamp is None:
            timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        directory = os.path.dirname(filepath)
        filename = os.path.basename(filepath)
        if directory:
            os.makedirs(directory, exist_ok=True)
        filename_with_timestamp = os.path.join(directory, f"{filename}_{timestamp}.csv")
        pandas_df = df.toPandas()
        pandas_df.to_csv(filename_with_timestamp, index=False)
        print(f"Successfully saved to {filename_with_timestamp}")
```

```
        return filename_with_timestamp
    except Exception as e:
        print(f"Error saving to CSV: {str(e)}")
        return None


def calculate_longest_streaks(data):
    """get the longest streak for each location.
    @ param:   df with streak information
    @ returns: df with longest streaks"""
    # filter for good aqi (1-3)
    data_good = data.filter(col("AQI").between(1, 3))
    # convert 'date' column to datetype if not already
    data_good = data_good.withColumn("date", to_date("timestamp"))
    # define window partitioned by location and ordered by date
    window_spec = Window.partitionBy("latitude", "longitude").orderBy("date")
    # calculate the difference in days between consecutive dates
    data_good = data_good.withColumn("prev_date", lag("date").over(window_spec))
    data_good = data_good.withColumn(
        "date_diff",
        when(col("prev_date").isNull(), 1).otherwise(
            datediff(col("date"), col("prev_date"))
        ),
    )
    # identify breaks in streaks (date difference > 1 indicates a break)
    data_good = data_good.withColumn(
        "new_streak", when(col("date_diff") > 1, 1).otherwise(0)
    )
    # assign streak ids
    data_good = data_good.withColumn("streak_id", Fsum("new_streak").over(window_spec))
    # calculate streak lengths
    streak_lengths = data_good.groupBy("latitude", "longitude", "streak_id").agg(
        F.count("*").alias("streak_length")
    )
    # get the longest streak
    longest_streaks = streak_lengths.groupBy("latitude", "longitude").agg(
        F.max("streak_length").alias("max_streak_length")
    )
    return longest_streaks


def plot_streak_histogram_outliers(longest_streaks, output_dir, timestamp):
    """create histogram of air quality streaks.
    @ param:  longest_streaks: df with max_streak_length column
              timestamp: timestamp for filename
              include_outliers:
    @ return: Path to saved plot
    """
    longest_streaks = filter_invalid_coordinates(longest_streaks)
    plt.figure(figsize=(12, 6))
    streaks_pd = longest_streaks.select("max_streak_length").toPandas()
    max_streak = streaks_pd["max_streak_length"].max()
    bins = np.linspace(0, max_streak, num=21)
    plt.hist(
        streaks_pd["max_streak_length"],
        bins=bins,
        edgecolor="black",
        alpha=0.7,
        color="#5499C7",
    )
    mean_streak = streaks_pd["max_streak_length"].mean()
    median_streak = streaks_pd["max_streak_length"].median()
    plt.xlim(0, max_streak)
    plt.grid(True, alpha=0.3, linestyle="--")
    plt.title(
        "Distribution of Longest Good Air Quality Streaks\n"
```

```
        + "Data collected from 2024-11-22 20:09:13 to 2024-11-24 20:11:13",
        pad=20,
    )
    plt.xlabel("Streak Length", labelpad=10)
    plt.axvline(
        mean_streak,
        color="#E74C3C",
        linestyle="--",
        linewidth=2,
        label=f"Mean: {mean_streak:.1f}",
    )
    plt.axvline(
        median_streak,
        color="#27AE60",
        linestyle="--",
        linewidth=2,
        label=f"Median: {median_streak:.1f}",
    )
    plt.legend(framealpha=0.5, fancybox=True, shadow=True)
    plt.tight_layout()
    filename = f"streak_distribution_woutliers_{timestamp}.png"
    filepath = os.path.join(output_dir, filename)
    plt.savefig(filepath, dpi=300, bbox_inches="tight", facecolor="white")
    plt.close()
    return filepath


def plot_streak_histogram(longest_streaks, output_dir, timestamp):
    """create histogram of air quality streaks.
    @ param:  longest_streaks: df with max_streak_length column
              timestamp: timestamp for filename
              include_outliers:
    @ return: Path to saved plot
    """
    longest_streaks = filter_invalid_coordinates(longest_streaks)
    plt.figure(figsize=(12, 6))
    streaks_pd = longest_streaks.select("max_streak_length").toPandas()
    Q1 = streaks_pd["max_streak_length"].quantile(0.25)
    Q3 = streaks_pd["max_streak_length"].quantile(0.75)
    IQR = Q3 - Q1
    upper_bound = Q3 + 1.5 * IQR
    # filter out outliers
    filtered_streaks = streaks_pd[streaks_pd["max_streak_length"] <= upper_bound]
    max_streak = filtered_streaks["max_streak_length"].max()
    bins = np.linspace(0, max_streak, num=21)
    plt.hist(
        filtered_streaks["max_streak_length"],
        bins=bins,
        edgecolor="black",
        alpha=0.7,
        color="#5499C7",
    )
    mean_streak = filtered_streaks["max_streak_length"].mean()
    median_streak = filtered_streaks["max_streak_length"].median()
    plt.xlim(0, max_streak)
    plt.ylim(bottom=0)
    plt.grid(True, alpha=0.3, linestyle="--")
    plt.title(
        "Distribution of Longest Good Air Quality Streaks\n"
        + "Data collected from 2024-11-22 02:09:13 to 2024-11-24 02:22:11\n",
        pad=20,
    )
    plt.xlabel("Streak Length", labelpad=10)
    plt.axvline(
        mean_streak,
        color="#E74C3C",
```

```
        linestyle="--",
        linewidth=2,
        label=f"Mean: {mean_streak:.1f}",
    )
    plt.axvline(
        median_streak,
        color="#27AE60",
        linestyle="--",
        linewidth=2,
        label=f"Median: {median_streak:.1f}",
    )
    plt.legend(framealpha=0.5, fancybox=True, shadow=True)
    plt.tight_layout()
    filename = f"streak_distribution_{timestamp}.png"
    filepath = os.path.join(output_dir, filename)
    plt.savefig(filepath, dpi=300, bbox_inches="tight", facecolor="white")
    plt.close()
    n_outliers = len(streaks_pd) - len(filtered_streaks)
    print(f"Removed {n_outliers} outliers (streaks > {upper_bound:.1f})")
    return filepath


def calculate_wcss(model):
    """calculate within cluster sum of squares
    @ param:  kmeans model object
    @ return: float (wcss)
    """
    summary = model.summary
    wcss = summary.trainingCost
    return wcss


def find_elbow_point(k_values, wcss_values):
    """find optimal k using elbow method
    @ param:  k_values: list of k values tested
              wcss_values: list of wcss scores for each k
    @ return: optimal k value
    """
    k_array = np.array(k_values)
    wcss_array = np.array(wcss_values)
    k_norm = (k_array - k_array.min()) / (k_array.max() - k_array.min())
    wcss_norm = (wcss_array - wcss_array.min()) / (wcss_array.max() - wcss_array.min())
    gradients = np.gradient(wcss_norm, k_norm)
    second_derivatives = np.gradient(gradients, k_norm)
    elbow_idx = np.argmax(np.abs(second_derivatives))
    optimal_k = k_values[elbow_idx]
    return optimal_k


def find_optimal_k(df, k_range=(40, 100), step=10):
    """find best k for clustering using elbow method.
    @ param:  df: df with lat/long columns
              k_range: tuple of (min_k, max_k) to test
              step: increment between k values
    @ return: tuple of (optimal_k, wcss_values, k_values)
    """
    df = df.withColumn(
        "scaled_longitude", F.col("longitude") * F.cos(F.radians(F.col("latitude")))
    )
    assembler = VectorAssembler(
        inputCols=["latitude", "scaled_longitude"], outputCol="features"
    )
    df_assembled = assembler.transform(df)
    k_values = list(range(k_range[0], k_range[1] + 1, step))
    wcss_values = []
    for k in k_values:
```

```
        kmeans = KMeans().setK(k).setSeed(1)
        model = kmeans.fit(df_assembled)
        wcss = calculate_wcss(model)
        wcss_values.append(wcss)
        print(f"Completed clustering with k={k}, WCSS={wcss:.2f}")
    # find the elbow point
    optimal_k = find_elbow_point(k_values, wcss_values)
    # plot elbow curve
    plt.figure(figsize=(10, 6))
    plt.plot(k_values, wcss_values, "bo-")
    plt.plot(
        optimal_k,
        wcss_values[k_values.index(optimal_k)],
        "ro",
        label=f"Elbow Point (k={optimal_k})",
    )
    plt.xlabel("Number of Clusters (k)")
    plt.ylabel("Within-Cluster Sum of Squares (WCSS)")
    plt.title("Elbow Method for Optimal k Selection")
    plt.legend()
    plt.grid(True)
    # save the plot
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    plt.savefig(f"./output/elbow_curve_{timestamp}.png", dpi=300, bbox_inches="tight")
    plt.close()
    return optimal_k, wcss_values, k_values


def cluster_regions(df, k=None):
    """perform kmeans clustering
    @ param:  df: df with lat/long columns
    @ return: tuple of (clustered_df, cluster_centers)
    """
    if k is None:
        print("Finding optimal number of clusters...")
        optimal_k, _, _ = find_optimal_k(df)
        k = optimal_k
        print(f"Optimal number of clusters determined to be: {k}")
    df = df.withColumn(
        "scaled_longitude", F.col("longitude") * F.cos(F.radians(F.col("latitude")))
    )
    assembler = VectorAssembler(
        inputCols=["latitude", "scaled_longitude"], outputCol="features"
    )
    df = assembler.transform(df)
    # train kmeans model
    kmeans = KMeans().setK(k).setSeed(1)
    model = kmeans.fit(df)
    # predict clusters
    df = model.transform(df)
    df = df.withColumnRenamed("prediction", "cluster_id")
    return df, model.clusterCenters()


def save_plot(plt, filename, timestamp=None):
    """save matplotlib plot with timestamp.
    @ param:  plt
              filename: base name for file
              timestamp
    @ return: path to saved file or None if error
    """
    try:
        if timestamp is None:
            timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        os.makedirs("output", exist_ok=True)
        filename_with_timestamp = os.path.join("output", f"{filename}_{timestamp}.png")
```

```
        plt.savefig(filename_with_timestamp, dpi=300, bbox_inches="tight")
        print(f"Successfully saved plot to {filename_with_timestamp}")
        return filename_with_timestamp
    except Exception as e:
        print(f"Error saving plot: {str(e)}")
        return None


def main():
    spark = (
        SparkSession.builder.appName("Air Quality Analysis")
        .config("spark.master", "local[*]")
        .config("spark.hadoop.fs.defaultFS", "file:///")
        .getOrCreate()
    )
    spark.sparkContext.setLogLevel("ERROR")
    # get current timestamp for filenames
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    # set the directory where data is being (or to be) written
    data_dir_relative = "./Data/Streaming"
    data_dir_absolute = os.path.abspath(data_dir_relative)
    data_dir = data_dir_absolute
    os.makedirs(data_dir, exist_ok=True)
    accumulated_data_dir_relative = "./Data/accumulated_data"
    accumulated_data_dir_absolute = os.path.abspath(accumulated_data_dir_relative)
    accumulated_data_dir = accumulated_data_dir_absolute
    os.makedirs(accumulated_data_dir, exist_ok=True)
    # synchronous data collection
    data_collection_thread = threading.Thread(
        target=collect_data_over_time,
        args=(data_dir,),
        kwargs={"duration_hours": 0, "interval_minutes": 0},
    )
    data_collection_thread.start()
    print("Data collection thread started.")
    data_collection_thread.join()
    print("Data collection completed!")
    # schema definition
    sensor_schema = StructType(
        [
            StructField("value_type", StringType(), True),
            StructField("value", StringType(), True),
        ]
    )
    schema = StructType(
        [
            StructField("id", LongType(), True),
            StructField("sampling_rate", StringType(), True),
            StructField("timestamp", StringType(), True),
            StructField(
                "location",
                StructType(
                    [
                        StructField("id", LongType(), True),
                        StructField("latitude", StringType(), True),
                        StructField("longitude", StringType(), True),
                        StructField("altitude", StringType(), True),
                        StructField("country", StringType(), True),
                        StructField("exact_location", IntegerType(), True),
                        StructField("indoor", IntegerType(), True),
                    ]
                ),
            ),
            StructField(
                "sensor",
                StructType(
```

```
                    [
                        StructField("id", LongType(), True),
                        StructField("pin", StringType(), True),
                        StructField(
                            "sensor_type",
                            StructType(
                                [
                                    StructField("id", LongType(), True),
                                    StructField("name", StringType(), True),
                                    StructField("manufacturer", StringType(), True),
                                ]
                            ),
                        ),
                    ]
                ),
            ),
            StructField("sensordatavalues", ArrayType(sensor_schema), True),
        ]
    )
    current_df = spark.read.schema(schema).json(f"{data_dir}/*current_data*.json")
    # historical_df = spark.read.schema(schema).json(f"{data_dir}/*historical_data*.json")

    def process_batch(batch_df, batch_id):
        print(f"Processing batch {batch_id}")
        # process data, pivot data and calculate aqi
        current_data_processed = process_data(batch_df)
        current_pivot = pivot_data(current_data_processed)
        current_pivot = current_pivot.withColumn(
            "AQI", aqi_udf(F.col("P2"), F.col("P1"))
        )
        print(
            f"Final DataFrame size after AQI calculation: {current_pivot.count():,} records
"
        )
        current_pivot.write.mode("append").parquet(accumulated_data_dir)
        # read all accumulated data and cache the accumulated data
        accumulated_data = spark.read.parquet(accumulated_data_dir)
        accumulated_data.cache()

        # task 1: country-level analysis
        print("\nTask 1: Analyzing country-level air quality improvements...")
        country_aqi = current_pivot.groupBy("country", "date").agg(
            F.avg("AQI").alias("avg_aqi"), F.count("*").alias("num_sensors")
        )
        window_spec = Window.partitionBy("country").orderBy("date")
        country_aqi = country_aqi.withColumn(
            "prev_day_aqi", F.lag("avg_aqi").over(window_spec)
        )
        country_aqi = country_aqi.withColumn(
            "aqi_improvement", F.col("prev_day_aqi") - F.col("avg_aqi")
        )
        country_aqi = country_aqi.dropna(subset=["aqi_improvement"])
        max_date = country_aqi.select(F.max("date")).collect()[0][0]
        latest_country_aqi = country_aqi.filter(F.col("date") == max_date)
        top_countries = latest_country_aqi.orderBy(F.desc("aqi_improvement")).limit(10)
        print("\nTop 10 countries with greatest air quality improvement:")
        top_countries.show(truncate=False)
        save_to_csv(top_countries, "./Data/top_countries", timestamp)

        # task 2: regional analysis
        print("\nTask 2: Clustering regions and analyzing improvements...")
        clustered_data, cluster_centers = cluster_regions(current_pivot, k=None)
        cluster_centers_df = spark.createDataFrame(
            [
                (i, float(center[0]), float(center[1]))
                for i, center in enumerate(cluster_centers)
```

```
    ],
    ["cluster_id", "latitude", "unscaled_longitude"],
)
# unscale longitude (reverse the scaling done in cluster_regions)
cluster_centers_df = cluster_centers_df.withColumn(
    "longitude",
    F.col("unscaled_longitude") / F.cos(F.radians(F.col("latitude"))),
)
cluster_aqi = clustered_data.groupBy("cluster_id", "date").agg(
    F.avg("AQI").alias("avg_aqi"),
    F.count("*").alias("num_sensors"),
    F.avg("latitude").alias("center_lat"),
    F.avg("longitude").alias("center_lon"),
)
cluster_aqi = cluster_aqi.join(
    cluster_centers_df.select("cluster_id", "latitude", "longitude"),
    on="cluster_id",
)
window_spec = Window.partitionBy("cluster_id").orderBy("date")
cluster_aqi = cluster_aqi.withColumn(
    "prev_day_aqi", F.lag("avg_aqi").over(window_spec)
)
cluster_aqi = cluster_aqi.withColumn(
    "aqi_improvement", F.col("prev_day_aqi") - F.col("avg_aqi")
)
latest_cluster_aqi = cluster_aqi.filter(F.col("date") == max_date)
top_clusters = latest_cluster_aqi.orderBy(F.desc("aqi_improvement")).limit(50)
print("\nTop 50 regions with greatest air quality improvement:")
top_clusters.show(truncate=False)
save_to_csv(top_clusters, "./Data/top_regions", timestamp)

# task 3: calculating longest streaks of good air quality
print("\nTask 3: Calculating longest streaks of good air quality...")
longest_streaks = calculate_longest_streaks(accumulated_data)
# get distinct (latitude, longitude, country) mappings
location_country = accumulated_data.select(
    "latitude", "longitude", "country"
).distinct()
longest_streaks = longest_streaks.join(
    location_country, on=["latitude", "longitude"], how="left"
)
longest_streaks.cache()
save_to_csv(longest_streaks, "./Data/longest_streaks", timestamp)
histogram_path = plot_streak_histogram(longest_streaks, "./Data", timestamp)
histogram_path = plot_streak_histogram_outliers(
    longest_streaks, "./Data", timestamp
)
print(f"Created streak histogram at: {histogram_path}")
print("\nListing top 20 geographical locations of regions with AQI data...")
location_counts = accumulated_data.groupBy(
    "latitude", "longitude", "country"
).agg(F.count("*").alias("num_records"))
location_counts = filter_invalid_coordinates(location_counts)
top_locations = location_counts.orderBy(F.desc("num_records")).limit(20)
top_locations.show(truncate=False)
save_to_csv(top_locations, "./Data/top_locations", timestamp)

# Plotting geographical locations
print("\nPlotting geographical locations of regions...")
gdf = gpd.GeoDataFrame(
    current_pivot.select("latitude", "longitude").dropna().toPandas(),
    geometry=gpd.points_from_xy(
        current_pivot.select("longitude").dropna().toPandas()["longitude"],
        current_pivot.select("latitude").dropna().toPandas()["latitude"],
    ),
    crs="EPSG:4326",
```

```
        )
        # Create GeoDataFrame for cluster centers
        centers_pd = cluster_centers_df.select("latitude", "longitude").toPandas()
        centers_gdf = gpd.GeoDataFrame(
            centers_pd,
            geometry=gpd.points_from_xy(
                centers_pd["longitude"], centers_pd["latitude"]
            ),
            crs="EPSG:4326",
        )
        world = gpd.read_file("./Files/ne_110m_admin_0_countries.shp")
        fig, ax = plt.subplots(figsize=(18, 12))
        world.boundary.plot(ax=ax, linewidth=1)
        gdf.plot(
            ax=ax,
            marker="o",
            color="orange",
            markersize=5,
            alpha=0.6,
            label="AQI Data Points",
        )
        centers_gdf.plot(
            ax=ax,
            marker="*",
            color="red",
            markersize=100,
            alpha=0.8,
            label="Cluster Centers",
        )
        plt.title("Geographical Locations of Regions with AQI Data")
        plt.xlabel("Longitude")
        plt.ylabel("Latitude")
        plt.legend()
        plt.tight_layout()
        save_plot(plt, "geographic_distribution", timestamp)
        plt.close(fig)

        accumulated_data.unpersist()
        longest_streaks.unpersist()
    process_batch(current_df, batch_id=0)
    spark.stop()

if __name__ == "__main__":
    main()
```