# *Comparative Analysis of Carbon Intensity and Carbon-Free Energy in Italy and Sweden (2021-2024) Using Apache Spark*

Matteo Basili
*Department of Civil Engineering and Computer Engineering*
*University of Rome Tor Vergata*
Rome, Italy
matteo.basili@students.uniroma2.eu

Adriano Trani
*Department of Civil Engineering and Computer Engineering*
*University of Rome Tor Vergata*
Rome, Italy
adriano.trani@students.uniroma2.eu

***Abstract*—The project analyzes hourly data on direct carbon intensity and the share of carbon-free energy for Italy and Sweden over the 2021–2024 period, using Apache Spark in cluster mode via Docker Compose. Data ingestion was handled with Apache NiFi and stored in HDFS in both CSV and Parquet formats. Three queries were implemented in Python using RDDs, DataFrames, and Spark SQL, with an experimental comparison. Results were exported to Redis and visualized using Grafana. The analysis offers a quantitative comparison between the two countries and demonstrates the scalability of Spark for high-granularity time-series data.**

***Keywords—Apache Spark, PySpark, Spark SQL, RDD, DataFrame, Apache NiFi, Data Ingestion, HDFS, Redis, Grafana, Carbon Intensity Analysis, Carbon-Free Energy Percentage, Temporal Data Aggregation, Performance Benchmarking, Docker Compose, Italy Energy Data, Sweden Energy Data***

## I. Introduction

The analysis of carbon emissions and renewable energy production is crucial for the energy transition. This project processes high-resolution hourly data for **Italy** and **Sweden** (**2021–2024**), measuring **direct carbon intensity** ($gCO_2eq/kWh$) and the share of **carbon-free energy** (CFE%).

**Apache Spark** was selected to handle large-scale data, executed in cluster mode on a single physical node using **Docker Compose.** Data transformations were implemented in **PySpark**.

**Apache NiFi** managed the ingestion, preprocessing **Electricity Maps** datasets and storing them in **HDFS** (**CSV** and **Parquet**). Outputs were exported to **Redis** and visualized in **Grafana**.

The project addresses three main queries:

- **Q1 – Yearly aggregation by country**: average, minimum, and maximum of carbon intensity and CFE%, with comparative plots.
- **Q2 – Monthly aggregation for Italy**: top 5 and bottom 5 months for each metric, with visual trends.
- **Q3 – 24-hour aggregation**: for both countries, computation of minimum, 25th, 50th, and 75th percentiles and maximum values for the two metrics, with hourly comparison plots.

Each query was executed using both programmatic APIs (**RDD/DataFrame**) and **Spark SQL** to compare performance. Tests were run under controlled conditions and repeated to ensure statistically significant timing results.

## II. Background and Theory

### A. Apache Spark: Concepts and Programming Models

Apache Spark [1] is an open-source engine for distributed data processing, optimized for performance via in-memory computation.

It provides three main interfaces:

- **RDD (Resilient Distributed Dataset):** low-level structure with detailed control but more complex code.
- **DataFrame:** tabular distributed data with automatic optimizations.
- **Spark SQL:** enables SQL-like queries on DataFrames, useful for non-technical users and analytical queries.

All APIs use the same execution engine, enabling direct performance comparison.

### B. Electricity Maps Dataset: Structure and Semantics

Data comes from Electricity Maps [2], with hourly coverage for Italy and Sweden (2021–2024). Relevant fields include:

- **Datetime (UTC):** hourly timestamp.
- **Zone ID:** ISO country code (e.g., IT or SE).
- **Carbon Intensity $gCO_2eq/kWh$ (direct)**: direct emissions per kWh produced.
- **Carbon-Free Energy Percentage (CFE%):** share of electricity from carbon-free sources.

### C. Environmental Metrics: Meaning and Implications

- ***Carbon Intensity* –** measures GHG emissions per kWh; lower values mean cleaner energy production.

- ***Carbon-Free Energy Percentage (CFE%)*** **–** shows how much energy comes from emission-free sources. Higher values reflect greater use of clean technologies.

The combined analysis of both indicators provides a balanced view of a country's energy system's environmental efficiency.

## III. Solution Design

### A. General Architecture

The project adopts a **pipeline** architecture with four main phases: **Ingestion**, **Processing**, **Storage**, and **Visualization**. The entire system is containerized using Docker Compose [3], ensuring portability and reproducibility.

The main components are:

- ***Apache NiFi*** [4] for data flow management and transformation;
- ***HDFS*** [5] as a distributed storage for CSV and Parquet formats;
- ***Apache Spark*** [6] for parallel data processing;
- ***Redis*** [7] for result export;
- ***Grafana*** [8] for visualization.

Fig. 1 shows the six-stage data flow:

1. CSV files are retrieved from Electricity Maps via API.
2. NiFi preprocesses and loads data into HDFS (CSV and Parquet).
3. Spark reads data from HDFS for processing.
4. Query results are written back to HDFS in CSV format.
5. Aggregated results are exported to Redis.
6. Grafana reads the CSV data to visualize interactive dashboards.

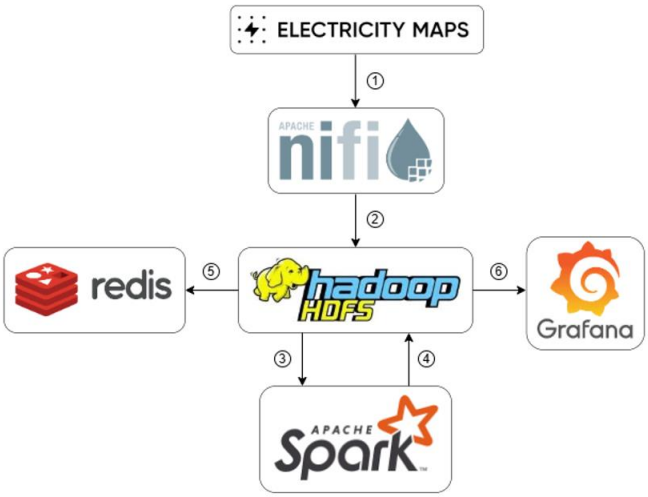


Fig. 1. Architecture of the system. *The arrows with numbers indicate the ordered data flow.*

### B. Data Ingestion with Apache NiFi

The ingestion pipeline included the following steps:

1. **Remote Ingestion** – Eight hourly CSV files (four per country, 2021–2024) were downloaded from Electricity Maps.
2. **Data Cleaning** – Only relevant fields were retained to reduce dataset size.
3. **Country-Level Consolidation** – Yearly files were merged into a single dataset per country.
4. **Dual Format Export** [9], [10]:
   i. ***CSV*** – a human-readable, interoperable format;
   ii. ***Parquet*** – a column binary format optimized for Spark.
5. **Load into HDFS** – Both CSV and Parquet versions were stored in HDFS.

This design offloads Spark from cleaning and normalization tasks, enabling efficient processing on already-structured data.

### C. Computation with Apache Spark

Two computational paradigms were adopted:

- **Programmatic APIs (RDD and DataFrame)** – for manual logic implementation;
- **Spark SQL** – for declarative query expression using SQL.

***Input Format Selection***

- **RDD → CSV:** Queries based on RDDs were executed starting from the CSV format. This row-oriented and textual format require explicit parsing.
- **DataFrame/Spark SQL → Parquet:** Queries implemented using DataFrame and SQL were based on the Parquet format. Parquet enables optimizations like predicate pushdown and column pruning.

***Experiments and Benchmarking***

Queries were implemented using both approaches and tested under controlled conditions:

- No background system processes;
- Spark caching disabled between runs;
- Only Spark job processing times were measured.

Each test was repeated several times, with mean and standard deviation calculated to ensure statistically valid comparisons.

### D. Result Export and Visualization

Aggregated results were exported from Spark to Redis and visualized using Grafana, allowing for dynamic dashboards and time-series comparisons of key environmental metrics.

## IV. Solution Details

All the project's code is located in a Github repository [11]. All the folders, files and code calls mentioned in this section (and also in the next ones) are located within this repository.

### A. Data Acquisition – Electricity Maps → Apache NiFi

The acquisition phase is implemented in NiFi through a mini-workflow composed of three processors:

- ***Trigger (Init)*** – A *GenerateFlowFile* processor triggers the workflow.
- ***URL Generation (GenerateDataURLs)*** – A *ExecuteScript* processor (Groovy):
  - Generates 8 URLs (4 for Italy, 4 for Sweden)
  - Creates a FlowFile for each URL with attributes *filename* and *download_url*
- ***Download (GetElectricityDataFiles)*** – An *InvokeHTTP* processor:
  - Downloads CSVs from the respective URLs
  - Replaces FlowFile content with the payload
  - Propagates *filename* for later use

Result: 8 FlowFiles containing raw datasets, labeled by origin and name.

### B. Preprocessing in NiFi

***Objective***: normalize schema, retain only relevant columns, group by country, and consolidate into a single file per country in CSV and Parquet formats.

- **Field Cleaning and Selection (CleanData)** – *UpdateRecord* with *CSVReader/Writer*. It keeps and renames:
  - *datetime* ← "Datetime (UTC)"
  - *country_code* ← "Zone id"
  - *carbon_intensity_direct* ← "Carbon intensity $gCO_2eq/kWh$ (direct)"
  - *cfe_percent* ← "Carbon-free energy percentage (CFE%)"
- **Routing by Country (RouteOnAttribute)** – Files split into two flows (IT_ → italy, SE_ → sweden)
- **Route Labeling (UpdateAttribute)** – Adds *route* attribute
- **Merging Annual Files (MergeRecord)** – Merges yearly files for each country (2021–2024)
- **CSV File Renaming (RenameCsvFile)**:
  - *italy_2021_2024_clean.csv*
  - *sweden_2021_2024_clean.csv*
- **Parquet Conversion (ConvertToParquet)** – Converts CSVs to Parquet format
- **Parquet File Renaming (RenameParquetFile)**:
  - *italy_2021_2024_clean.parquet*
  - *sweden_2021_2024_clean.parquet*

### C. Persistence in HDFS – Apache NiFi → HDFS

The final datasets (in CSV and Parquet formats) are transferred to HDFS using a configured *PutHDFS* processor.

Files are written to the shared directory: *data/electricity*, and the final result is the presence of four files in the distributed file system:

- *italy_2021_2024_clean.csv*
- *sweden_2021_2024_clean.csv*
- *italy_2021_2024_clean.parquet*
- *sweden_2021_2024_clean.parquet*

Fig. 2 shows a screenshot of the complete dataflow as implemented in the NiFi GUI.

### D. Elaboration with Apache Spark – HDFS → Spark

Once the data is in HDFS, it is processed with Spark using three paradigms: RDD, DataFrame, and SQL, for each query (Q1, Q2, Q3).

**Q1 – RDD**

- Load CSVs with *textFile()*, parse into (year, country) keys.
- Use *reduceByKey()* to aggregate carbon, CFE, and stats (min/max).
- Compute averages and round results.

**Q1 – DataFrame**

- Read and merge Parquet files, convert types and extract year.
- Group by year and country; compute avg/min/max of carbon and CFE.
- Round results.

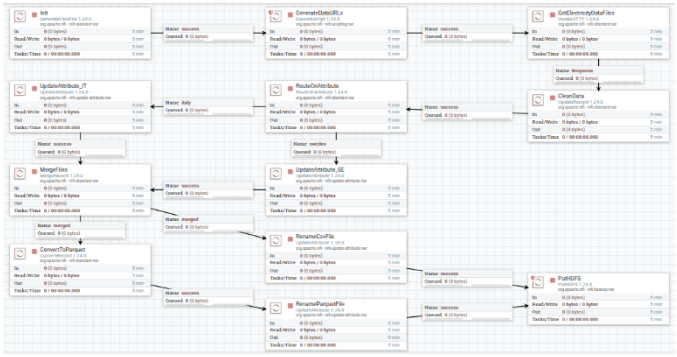

Fig. 2. Complete NiFi Dataflow. *Screenshot of the data pipeline from Electricity Maps to HDFS, with routing and preprocessing logic shown graphically.*

**Q1 – SQL**

- Register Parquet data as a temporary view.
- SQL query extracts year, groups data, and computes stats with rounding.

**Q2 – RDD**

- Read Italian CSV, parse to (year-month, carbon, CFE).
- Aggregate monthly with *reduceByKey()*, compute averages.
- Locally compute top 5 rankings for carbon and CFE (asc/desc).
- Format output with ranks and labels.

**Q2 – DataFrame**

- Load Italian Parquet; extract month, carbon, CFE as doubles.
- Group by month and compute averages; cache results.
- Define Windows and use *row_number()* to get top 5 per category.
- Merge all rankings.

**Q2 – SQL**

- Read and cache Italian data as SQL view.
- Monthly aggregation with *date_format()* and *trunc()*.
- Use ROW_NUMBER() for top 5 rankings in 4 categories.
- Combine results with UNION and sort.

**Q3 – RDD**

- Read CSVs, parse rows with error handling.
- Map by (hour, country), aggregate with *reduceByKey()*.
- Compute hourly averages; collect and calculate percentiles manually.

**Q3 – DataFrame**

- Load Parquet files, extract hour, cast columns.
- Group by hour and country for averages.
- Use *percentile()* to calculate min, 25th, median, 75th, max.

**Q3 – SQL**

- Register Italian and Swedish views; preprocess with SQL.
- Merge with UNION ALL, compute hourly averages.
- Use SQL for percentile metrics by country and variable.
- Clean cache.

*E. Writing Results – Spark → HDFS*

Each query's result is written to HDFS immediately after computation. Despite minor differences by paradigm (RDD, DataFrame, SQL), the process is consistent:

- Save to a unique timestamped path in */output/*
- Write as a CSV file with header
- Let Spark handle partitioning and avoid overwriting past runs

*F. Exporting Results – HDFS → Redis*

Three Python scripts (one per query) export the results to Redis:

- **Access WebHDFS –** Connect using *InsecureClient*, read path passed via CLI
- **Scan part- Files* –** Filter Spark output files, read with streaming
- **Parse Lines**
  - Ignore headers, split rows, validate fields, build keys:
    - Q1 → ***key****: Q1:<country>:<year>*
    - Q2 → ***key****: Q2:IT:<order_type>:<rank>*
    - Q3 → ***key****: Q3:<country>:<metric>*
- **Batch Writing to Redis –** Use non-transactional pipeline for fast HSET operations

*G. Visualizing Results – HDFS → Grafana*

A Python script automates dashboard generation and updates:

- **Create the CSV Data Source –** Use Grafana API and *marcusolsson-csv-datasource* plugin
- **Create the Dashboard and Panel –** Define and configure time series panel via API
- **Login and Save via Selenium [12] –** Use headless Chrome to save dashboard (workaround for API limits)
- **Render PNGs –** Call Grafana render service to export panel images
- **Update for Multiple Metrics –** Loop through metrics (e.g., Carbon Intensity, CFE) and generate respective PNGs

*H. Query Performance Measurement*

The *run_query_isolated.py* script benchmarks query performance under clean conditions:

- **Execution Logic –** Full stack shutdown and minimal service restart before each run:
  - Ensures no cache or process interference
- **Spark Job Execution –** Run via *spark-submit* in *spark-master* container

- Time measured before and after job

- **Repeat and Analyze –** Run 10 times, compute mean and standard deviation
  - Save report in: *Results/analysis/performance_<query>_<mode>_stats.txt*

**Design Rationale:**

- Full reset guarantees cold-start conditions
- NiFi is shut down to eliminate JVM impact
- Host-side timing captures realistic end-to-end job duration

## V. Deployment

The system is fully containerized with Docker and orchestrated via *docker-compose.yml*, defining all components required for processing and visualization.

**Core Containers**:

- **Hadoop HDFS** – One NameNode and two DataNodes (Apache Hadoop 3.3.6)
- **Apache NiFi** – Version 1.24.0, connected to HDFS with mounted configs and custom scripts
- **Apache Spark** – Spark cluster composed of one master and two workers, based on the Bitnami Spark 3.5.1 image.
- **Grafana & Renderer** –
  - Grafana 12.0.1 with CSV datasource plugin
  - grafana-image-renderer for exporting PNG charts
- **Redis** – In-memory store (Redis 7.2) for intermediate result export
- **Results Exporter** – Custom container to move results from HDFS to Redis

**Networking & Storage**

- Shared Docker bridge network: *sabd_network*
- Persistent HDFS storage via named volumes: *namenode_data*, *datanode1_data*, *datanode2_data*
- Shared volume *Results/csv* allows access to CSVs by both Hadoop and Grafana

### A. System Startup and Management

The system is launched with:

*docker compose up –d*

- NiFi available on port 8080
- Grafana accessible via port 3000
- *results_exporter* auto-starts to handle Redis export

### B. Setup and Execution

Pipeline automation is handled by the *run_full_pipeline.py* script, covering all phases: ingestion, processing, export, and visualization.

Further configuration, requirements, and usage details are provided in the GitHub repository README [11].

## VI. Results

This section presents the results produced by the system in an organized manner: the output files of the queries, the result charts for visual analysis, and summary data on the processing times of the jobs. All results are shown in the form of figures, while interpretation and discussion are deferred to the next section.

### A. Query Results

| year | country | carbon-mean | carbon-min | carbon-max | cfe-mean | cfe-min | cfe-max |
|---|---|---|---|---|---|---|---|
| 2021 | IT | 280.084245 | 121.24 | 439.06 | 46.305932 | 15.41 | 77.02 |
| 2022 | IT | 321.617976 | 121.38 | 447.33 | 41.244127 | 13.93 | 77.44 |
| 2023 | IT | 251.819465 | 74.44 | 429.93 | 51.596057 | 20.39 | 85.02 |
| 2024 | IT | 207.299189 | 50.18 | 345.65 | 57.431828 | 20.9 | 90.26 |
| 2021 | SE | 5.946325 | 1.5 | 55.07 | 98.962411 | 92.8 | 99.65 |
| 2022 | SE | 3.875823 | 0.54 | 50.58 | 99.551723 | 94.16 | 99.97 |
| 2023 | SE | 3.903308 | 0.31 | 51.57 | 99.525599 | 94.12 | 99.98 |
| 2024 | SE | 3.726149 | 0.25 | 45.61 | 99.557928 | 94.27 | 99.98 |

Fig. 3. Final Q1 output CSV.

| date | carbon-intensity | cfe | order_type | rank |
|---|---|---|---|---|
| 2022-12 | 360.52 | 35.83832 | carbon_desc | 1 |
| 2022-03 | 347.359073 | 35.822218 | carbon_desc | 2 |
| 2021-11 | 346.728514 | 33.076681 | carbon_desc | 3 |
| 2022-10 | 335.784745 | 39.167164 | carbon_desc | 4 |
| 2022-02 | 330.489896 | 38.980595 | carbon_desc | 5 |
| 2024-05 | 158.240887 | 68.989731 | carbon_asc | 1 |
| 2024-04 | 170.670889 | 66.253958 | carbon_asc | 2 |
| 2024-06 | 171.978792 | 65.487792 | carbon_asc | 3 |
| 2024-03 | 192.853871 | 60.919556 | carbon_asc | 4 |
| 2024-07 | 200.595995 | 57.939099 | carbon_asc | 5 |
| 2024-05 | 158.240887 | 68.989731 | cfe_desc | 1 |
| 2024-04 | 170.670889 | 66.253958 | cfe_desc | 2 |
| 2024-06 | 171.978792 | 65.487792 | cfe_desc | 3 |
| 2024-03 | 192.853871 | 60.919556 | cfe_desc | 4 |
| 2023-05 | 203.494489 | 59.877003 | cfe_desc | 5 |
| 2021-11 | 346.728514 | 33.076681 | cfe_asc | 1 |
| 2022-03 | 347.359073 | 35.822218 | cfe_asc | 2 |
| 2022-12 | 360.52 | 35.83832 | cfe_asc | 3 |
| 2022-01 | 326.947876 | 36.603683 | cfe_asc | 4 |
| 2021-12 | 329.303508 | 37.868817 | cfe_asc | 5 |

Fig. 4. Final Q2 output CSV.

| country | data | min | 25-perc | 50-perc | 75-perc | max |
|---|---|---|---|---|---|---|
| IT | carbon-intensity | 219.029329 | 241.060318 | 279.202916 | 285.008504 | 296.746208 |
| IT | cfe | 42.203176 | 45.728436 | 47.60011 | 53.149181 | 57.423648 |
| SE | carbon-intensity | 3.150062 | 3.765761 | 4.293638 | 4.876138 | 5.94718 |
| SE | cfe | 99.213936 | 99.338007 | 99.411328 | 99.472495 | 99.540979 |

Fig. 5. Final Q3 output CSV.

## B. Charts

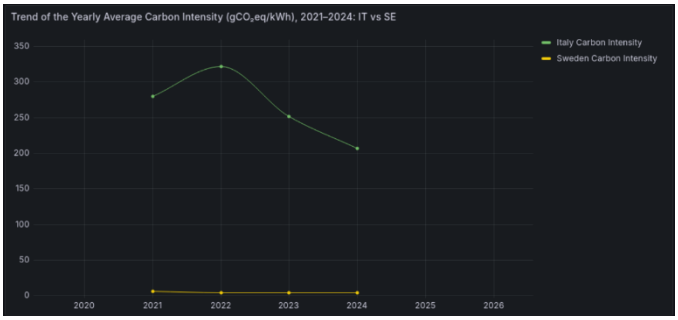


Fig. 6. Time series of carbon intensity results for Q1.

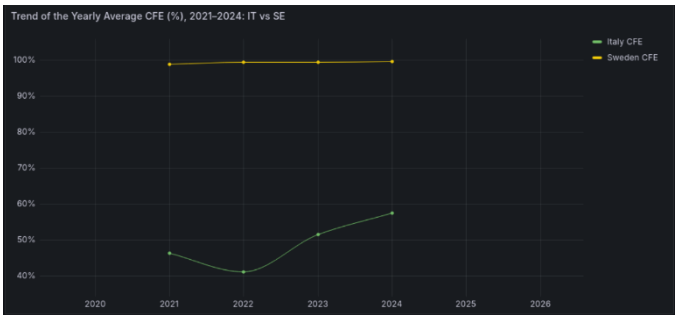


Fig. 7. Time series of CFE results for Q1.

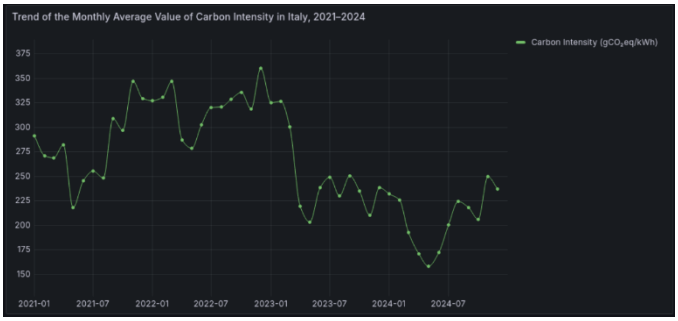


Fig. 8. Time series of carbon intensity results for Q2.

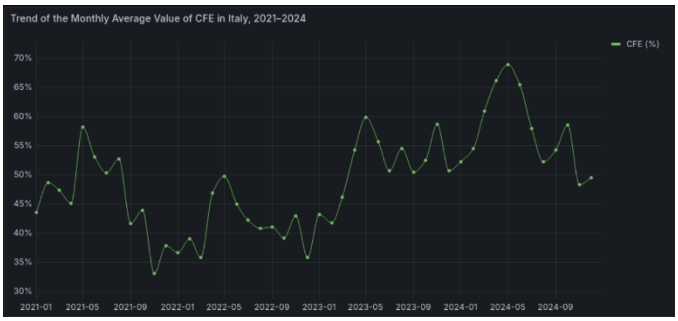


Fig. 9. Time series of CFE results for Q2.

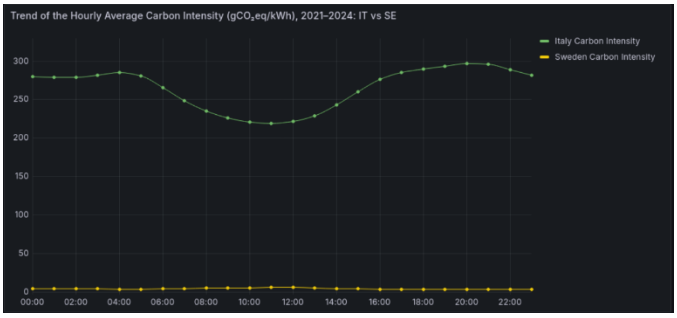


Fig. 10. Time series of carbon intensity results for Q3.

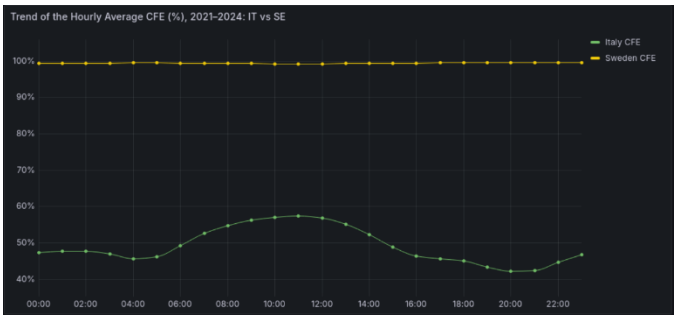


Fig. 11. Time series of CFE results for Q3.

## C. Processing Time Results

| Q1 | Average | Standard Deviation |
|---|---|---|
| RDD | 42.12 seconds | 3.05 seconds |
| DataFrame | 52.45 seconds | 11.07 seconds |
| SQL | 49.04 seconds | 0.73 seconds |

| Q2 | Average | Standard Deviation |
|---|---|---|
| RDD | 44.04 seconds | 0.59 seconds |
| DataFrame | 71.80 seconds | 1.12 seconds |
| SQL | 52.90 seconds | 1.43 seconds |

| Q3 | Average | Standard Deviation |
|---|---|---|
| RDD | 47.93 seconds | 1.13 seconds |
| DataFrame | 93.35 seconds | 1.60 seconds |
| SQL | 79.46 seconds | 0.81 seconds |

Fig. 12. Mean and standard deviation of execution times for queries Q1, Q2, and Q3 using Spark APIs: RDD, DataFrame, and SQL.

## VII. Discussion

This section analyzes the numerical and graphical results from Chapter VI, focusing on trends in carbon intensity and CFE %, differences between Italy and Sweden, and query performance.

### A. Annual Trend (Q1) – Italy vs. Sweden [Fig. 3, 6, 7]

**Italy**

- Carbon intensity dropped from ~322 to ~207 $gCO_2eq/kWh$ (–35%) between 2022–2024.
- CFE % increased from ~41% to ~57%.
- Improvement driven by more renewables, hydropower recovery, and low-carbon imports.

**Sweden**

- Stable near-zero carbon intensity (3–6 $gCO_2eq/kWh$) and ~99% CFE.
- Flat curves confirm a consistent, low-emission energy mix (hydro, nuclear, wind).

### B. Monthly Variability in Italy (Q2) – [Fig. 4, 8, 9]

- Peak carbon intensity: December 2022 (~360 $gCO_2eq/kWh$).
- Best month: May 2024 (~158 $gCO_2eq/kWh$). Inverse correlation with CFE % – low CFE coincides with high emissions and vice versa.
- 2024 dominates positive rankings, suggesting a structural improvement.

### C. Hourly Distribution (Q3) – [Fig. 5, 10, 11]

**Italy**

- Daily variation observed: lower carbon intensity at midday (~220 $gCO_2eq/kWh$), higher in evenings (~297).
- CFE % peaks during the day (~57%) and dips at night (~42%).

**Sweden**

- Extremely flat trends: carbon intensity <6 $gCO_2eq/kWh$ and stable CFE ~99.4–99.5%.
- Reflects a steady energy supply insensitive to daily demand changes.

### D. Computational Performance Analysis

The collected statistics (see Fig. 12) reveal significant differences among the three approaches in terms of average execution time and stability.

**Q1**

- ***RDD***: 42.12 s ($\sigma$ = 3.05 s)
- ***SQL***: 49.04 s ($\sigma$ = 0.73 s)
- ***DataFrame***: 52.45 s ($\sigma$ = 11.07 s)

→ RDD is fastest on flat aggregations; DataFrame shows high variability due to an outlier (~83 s).

**Q2**

- ***RDD***: 44.04 s ($\sigma$ = 0.59 s)
- ***SQL***: 52.90 s ($\sigma$ = 1.43 s)
- ***DataFrame***: 71.80 s ($\sigma$ = 1.12 s)
- → RDD remains stable and fastest, even with complex operations like window functions.

**Q3**

- ***RDD***: 47.93 s ($\sigma$ = 1.13 s)
- ***SQL***: 79.46 s ($\sigma$ = 0.81 s)
- ***DataFrame***: 93.35 s ($\sigma$ = 1.60 s)
- → Most complex query. RDD is significantly more performant, while DataFrame nearly doubles execution time.

These experimental executions were carried out on a Spark cluster using the default configuration without any manual tuning. The main specifications are:

- **Executors**: two, one per worker node;
- **Cores per executor**: 2 cores each, for a total of 4 available vCPUs;
- **Memory**: approximately 1 GB allocated per executor and driver;
- **Partitioning**: automatically determined based on HDFS block size (128 MB) for both CSV and Parquet files;
- **Deploy mode**: client mode, with the driver running on the spark-master node.

Although DataFrames and SQL queries on Parquet files are generally considered more efficient due to Spark optimizations (Catalyst, Tungsten) and the columnar format, results show that the RDD approach can be faster in simple scenarios with small datasets. This is because RDDs avoid the overhead related to serialization, schema-based parsing, and query planning, working in a more direct and linear way, especially on CSV files.

The Parquet format offers significant advantages in terms of reduced I/O and optimizations, often performing better on large datasets and complex operations. However, when data volume is limited or transformations are relatively simple, the overhead of schema management and optimization can make the benefits of Parquet less apparent or even reversed compared to CSV.

Ultimately, the efficiency of different solutions depends on multiple factors: dataset size, operation complexity, cluster configuration, parallelization, and caching. Therefore, it is important to evaluate on a case-by-case basis which approach is best suited, balancing performance and complexity.

### E. Some Other Considerations

Several challenges emerged during the project, especially regarding data and technology integration.

- Integrating NiFi, HDFS, Spark, Redis, and Grafana required deep understanding and REST API automation.

- Managing service permissions was simplified by running all containers as **root**—acceptable for development, but not for production.
- Despite initial complexity, the system proved **scalable** and **reproducible**, supporting automated pipelines and dynamic Grafana-based visualization.

## VIII. Conclusions

The project resulted in the development of a complete infrastructure for processing, analyzing, and visualizing complex energy data, integrating tools such as NiFi, HDFS, Spark, Redis, and Grafana.

The comparative performance analysis between RDD, DataFrame, and Spark SQL showed that, in the examined context, the RDD approach delivered the best execution times.

The project can be extended through the following optional activities:

- **Spark Performance Analysis:** Deepen the study of Spark's performance by modifying key configuration parameters (e.g., number of executors or worker nodes).
- **Task Q4 – Clustering Analysis:** Perform clustering on Carbon Intensity data ($gCO_2eq/kWh$, direct), aggregated annually and specifically for the year 2024.
    - *Goal*: identify groups of countries with similar emission patterns using the k-means algorithm (either implemented from scratch or via Spark MLlib [13]).
    - Determine the optimal number of clusters (k) using methods such as the elbow method [14] or the silhouette index [15].
    - The analysis will include 30 countries (15 European and 15 non-European) and will conclude with a visual chart showing the resulting cluster partitioning.

## References


[1] Apache Spark, "What is Apache Spark™?" Available online: https://spark.apache.org/

[2] Electricity Maps, "Datasets." Available online: https://portal.electricitymaps.com/datasets

[3] Docker Compose, "Documentation." Available online: https://docs.docker.com/compose/

[4] Apache Nifi, "Documentation." Available online: https://nifi.apache.org/documentation/v1/

[5] Apache Hadoop, "HDFS Users Guide." Available online: https://hadoop.apache.org/docs/r3.3.6/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html

[6] Apache Spark, "Documentation." Available online: https://archive.apache.org/dist/spark/docs/3.5.1/api/python/

[7] Redis, "Documentation." Available online: https://redis.io/docs/latest/

[8] Grafana, "Documentation." Available online: https://grafana.com/docs/

[9] Wikipedia, "Comma-separated values." Available online: https://it.wikipedia.org/wiki/Comma-separated_values

[10] Databricks, "What is Parquet?" Available online: https://www.databricks.com/glossary/what-is-parquet

[11] MatteoBasili, "sabd-progetto1-2024_25" [GitHub Repository], 2025. Available online: https://github.com/MatteoBasili/sabd-progetto1-2024_25

[12] Selenium, "Documentation." Available online: https://www.selenium.dev/documentation/

[13] Apache Spark, "MLlib." Available online: https://spark.apache.org/mllib/

[14] Wikipedia, "Elbow method (clustering)." Available online: https://en.wikipedia.org/wiki/Elbow_method_(clustering)

[15] Wikipedia, "Silhouette (clustering)." Available online: https://en.wikipedia.org/wiki/Silhouette_(clustering)