University of Luxembourg

Climate Insights

Time Series Forecasting with Apache Spark and Google Cloud

Bacem ETTEIB
Supvervised by: Prof. DI TOLLO

Motivation

- The problem:

1. ***Data Size Challenge:***

    - As datasets grow exponentially, processing them becomes a bottleneck for traditional tools like Pandas.
    - Inefficient handling of large datasets hampers the scalability and speed of data analysis.

2. ***Hardware Limitations:***

    - Limited memory capacity prevents loading and processing entire datasets at once.

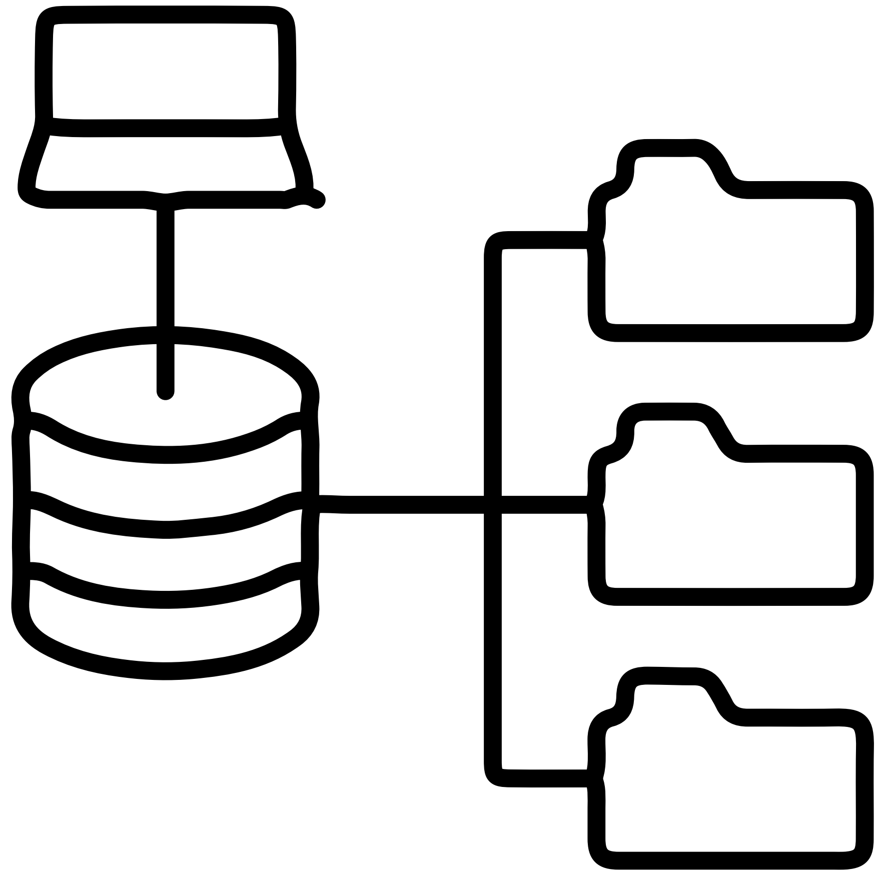
Motivation

- The solution:

  Distributed Processing System:

  - The optimal approach would be to implement a distributed processing system that can parallelize computations across multiple nodes, namely **Apache Spark framework.**

  - This technique facilitates optimal utilization of computing resources, resulting in high-speed data processing.



Motivation

- Strategy:

<u>Choosing Google Cloud</u>

- Google Cloud provides scalable solutions for distributed processing.

<u>Apache Spark Framework:</u>

- Apache Spark is chosen for its suitability for parallel processing.

<u>Demonstration Dataset:</u>

- Utilizing an open-source meteorological dataset for French regions.

THE APACHE® SOFTWARE FOUNDATION

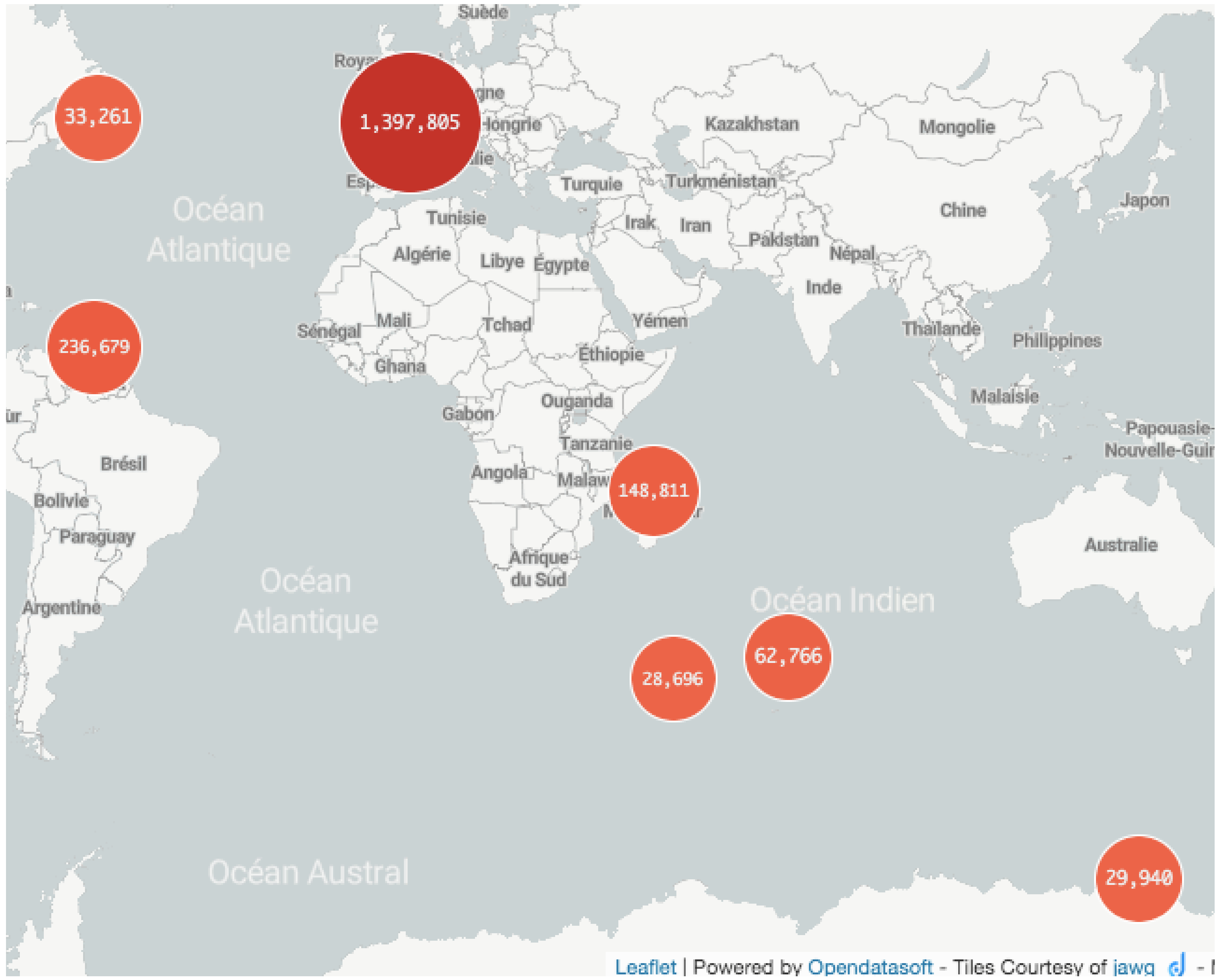
Requirements (1/2) - Large Dataset

1. ***Source:***

   - Open-source meteorological dataset representing French regions.
   - Provided by "Meteo France" and is freely available for download at **OpenDataSoft.**

2. ***Characteristics:***

   - The dataset comprises 2.1 gigabytes of historical weather data.
   - There are various data formats to choose from, including JSON and CSV.



Requirements (2/2) - Technical Stack

Skillset Expectation:

- Statistics and Data Analysis:

  Basic understanding required for effective data processing and interpretation.

- Python proficiency:

  Strong background in Python, preferably data analysis packages such as Pandas and Pyspark.

- Weather science:

  The project does not require specific advanced knowledge in weather science.

Overview (1/3) - Project Overview

1. ***Duration:***
   - A comprehensive scientific and technical report spanning six months.
2. ***Scope:***
   - Extensive research about distributed processing systems and time series forecasting using Spark.
3. ***Key Technologies:***
   - Apache Spark and Google Cloud to simulate a multi-cluster environment.
   - Dash to showcase the final analysis and forecasting results.

Overview (2/3) - Scientific Deliverable

Times Series Analysis:

In-depth exploration of several time series analysis techniques applied to the meteorological dataset.

Apache Spark Integration:

Clear explanation of distributed processing systems, Apache Spark architecture and its integration in the project.

Google Cloud:

Understanding of Google Cloud infrastructure for parallel computing.

Overview (3/3) - Technical Deliverable

Codebase and Results:

Highlights from the Python codebase, focusing on Pandas, PySpark, and EDA results.

Dashboard Creation and Demo:

Showcase of the interactive dashboard developed for weather analysis.

Challenges and Solutions

Discussion of challenges faced during the project and corresponding solutions.

University of Luxembourg

Problem statement

The study of the project through its scientific and technical aspects focuses on answering the following scientific question:

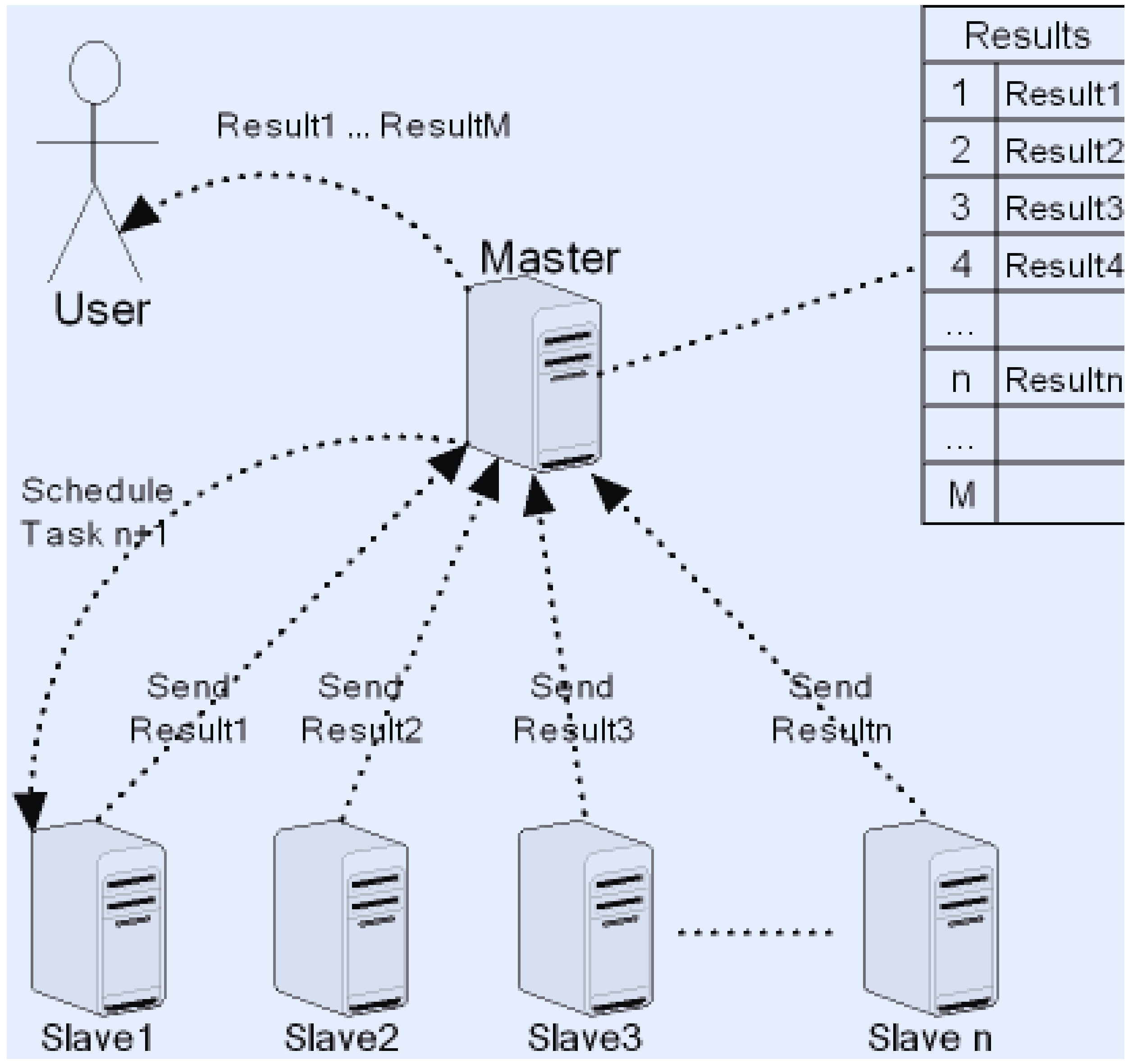
How can we combine Apache Spark with Time Series analysis for accurate meteorological data forecasting?

University of Luxembourg

Scientific Deliverable

Explanation of the design and production of the scientifc work.

Distributed Processing Systems



Distributed processing systems involve the use of multiple interconnected processors to collaboratively work on computational tasks. This paradigm enables parallel processing, dividing the workload among several machines.

The adoption of distributed processing in our project brings about improved performance as tasks can be executed concurrently, fault tolerance through redundancy, and scalability to handle large datasets or complex computations efficiently.

Apache Spark (1/2) - Overview

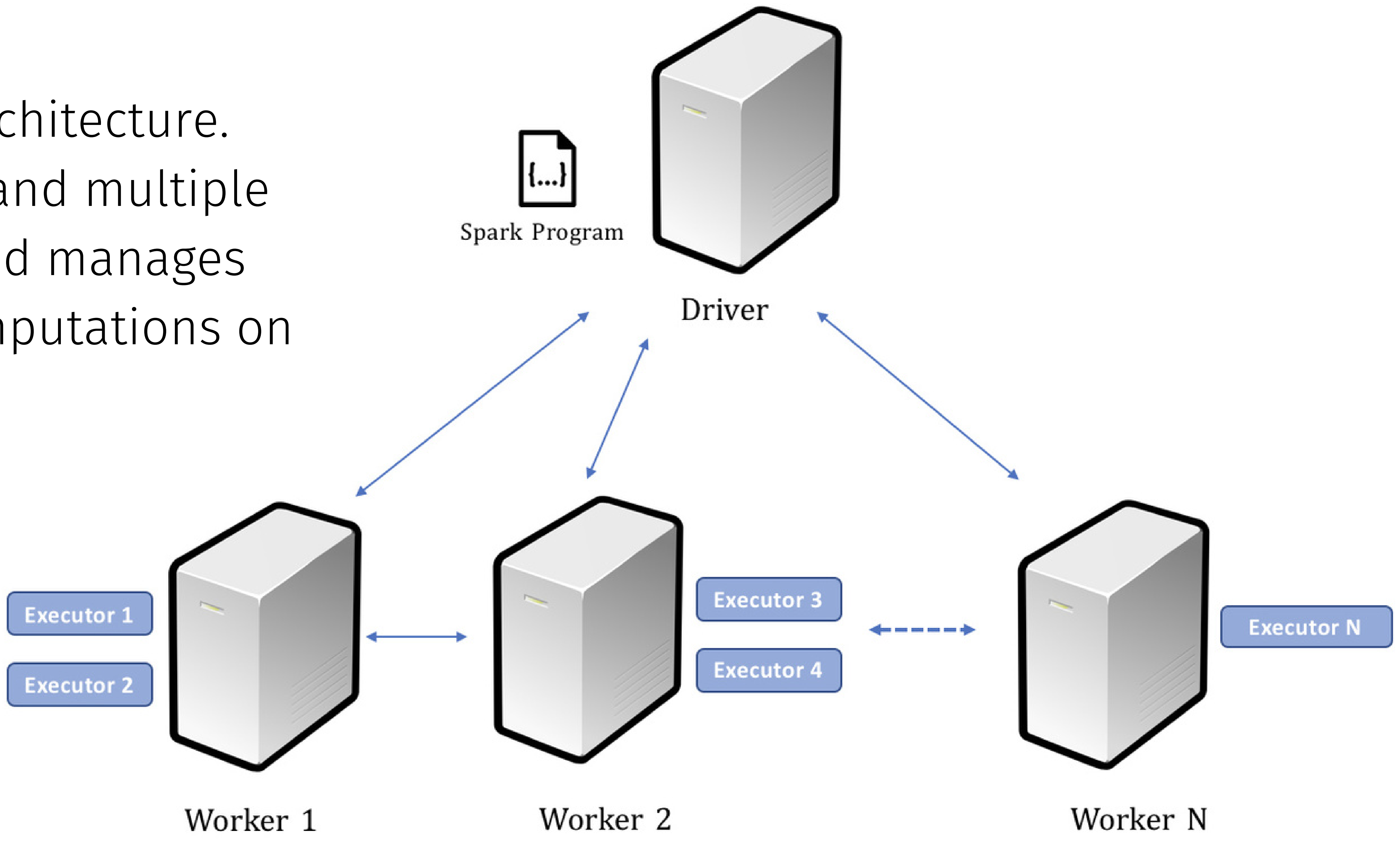
Apache Spark is a powerful open-source distributed computing system designed for high-speed, general-purpose cluster computing. It provides an expressive programming model and extensive libraries for diverse tasks, making it a versatile tool for data processing and analytics.

The architecture of Apache Spark is modular, with components interacting seamlessly to process data efficiently. This flexibility allows users to leverage the specific capabilities needed for their applications.

Apache Spark (2/2) - Architecture

- ***Master-Slave Architecture:***

Apache Spark follows a master-slave architecture. The cluster consists of a single master and multiple slaves. The master coordinates tasks and manages resources, while the slaves execute computations on the data.



Apache Spark (2/2) - Architecture

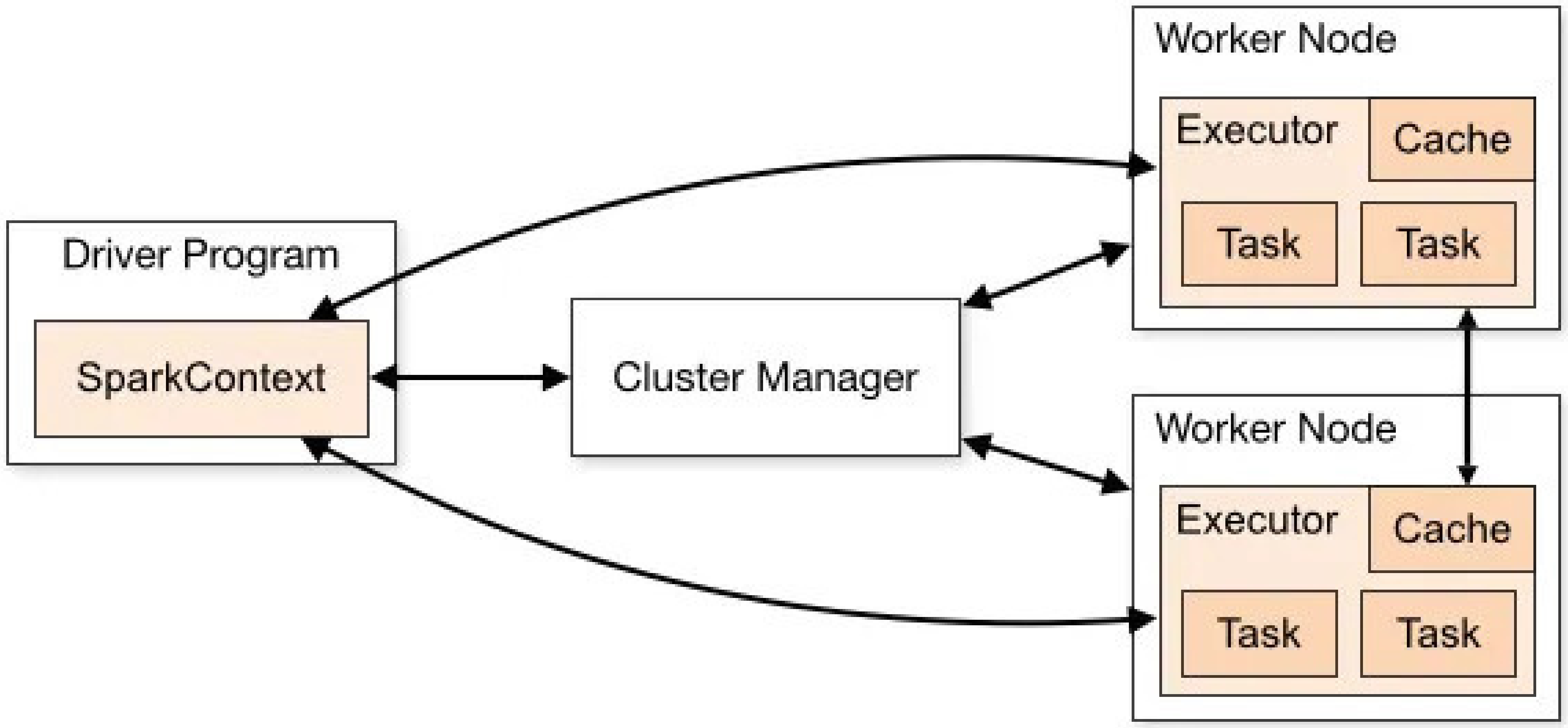
Spark consists of two main components:

- **Spark Driver:** where a user writes Spark code using Scala or Python. It is responsible for launching various parallel operations of the cluster.
- **Spark Executor:** the Java Virtual Machine (JVM) that runs on a worker node of the cluster. It provides hardware resources for running the tasks launched by the driver program.

The Cluster Manager allocates resources across applications. Spark supports various cluster managers like Hadoop YARN, Apache Mesos, and Standalone Scheduler.

Apache Spark (2/2) - Architecture

When a job is run, the driver parses the code, instructions are passed to the cluster manager to allocate resources and tasks are scheduled among worker nodes.



Time Series Analysis

Time series analysis is the examination of data points collected over time to identify patterns, trends, and anomalies.

- **Significance in Climate Data:**

  Timestamps in climate data enable nuanced exploration, revealing dynamic environmental conditions (e.g, pressure levels over time periods)

- **Crucial Techniques:**

  Statistical and machine learning techniques (trend analysis, seasonality detection, anomaly detection, and forecasting)

- **User-Defined Forecasting:**

  Our focus is on user-defined metric forecasting (e.g., temperature, pressure) using historical data patterns.

Time Series Analysis: Techniques

1. Seasonal Decomposition:

- Trend Analysis

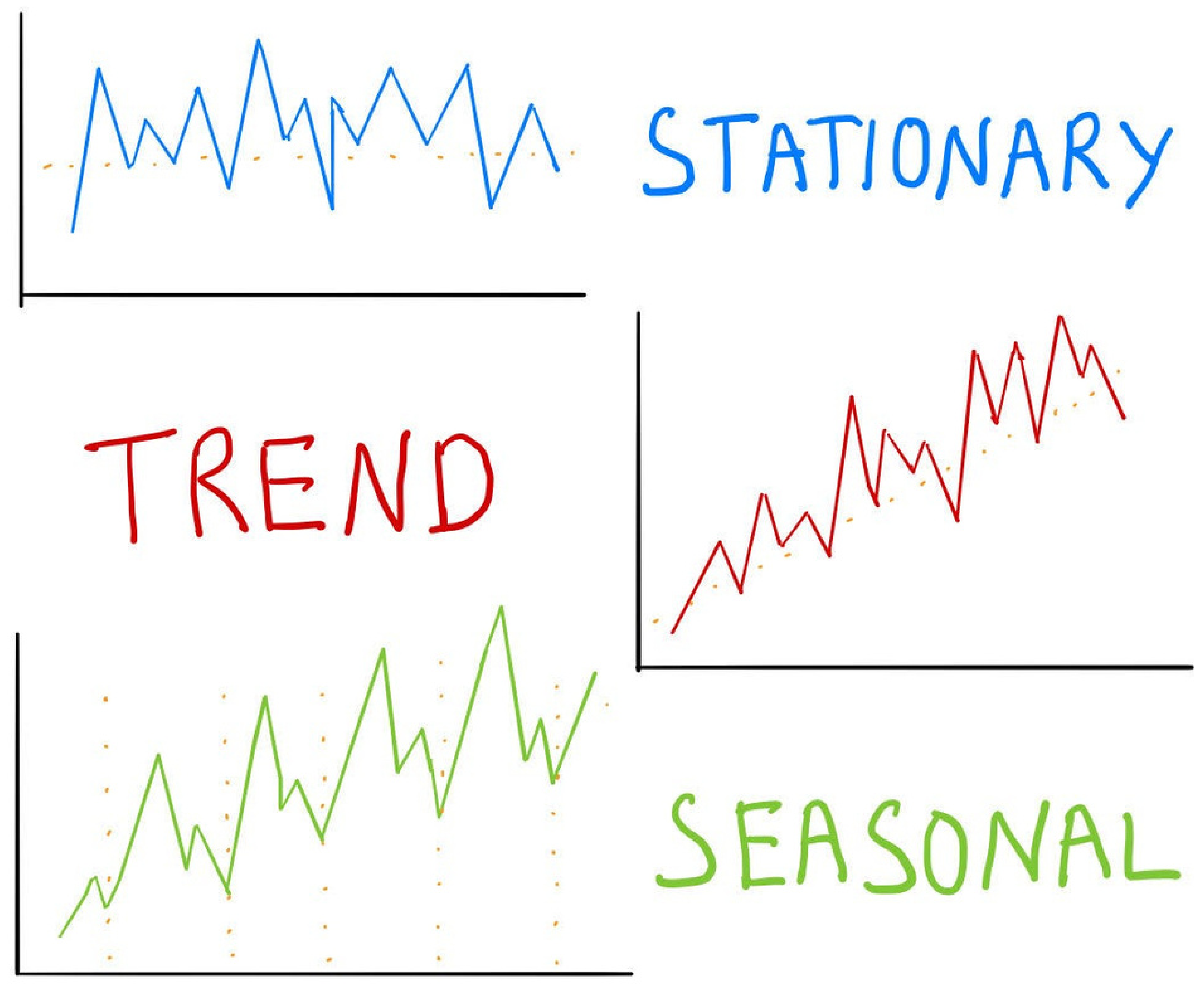
Captures the long-term movement or direction in the data.

- Seasonality:

Captures the regular, recurring patterns or seasonality in the data.

- Residual

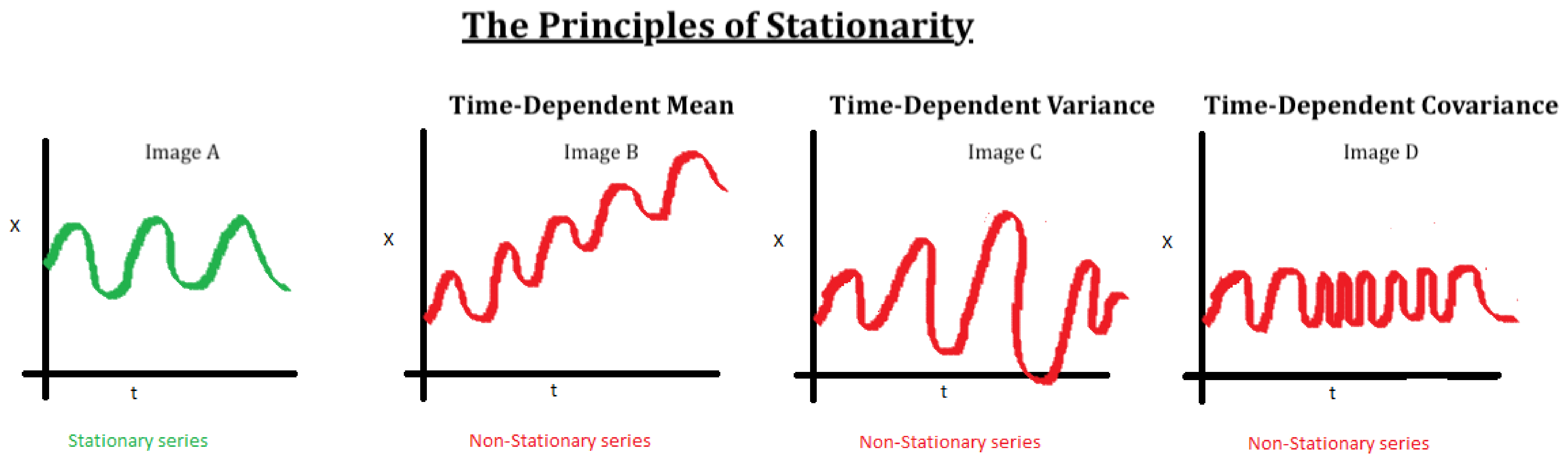
Represents the remaining variability after accounting for the components. It includes random fluctuations, measurement errors, or variations.



Time Series Analysis: Data Types

- Stationary data:

A stationary time series is one whose properties do not depend on the time at which the series is observed. Thus, time series with trends, or with seasonality, are not stationary — the trend and seasonality will affect the value of the time series at different times.



Time Series Analysis: Transformations

- Moving Average:

Moving Average (MA) is a statistical technique used in time series analysis to estimate the underlying trend or pattern in the data by averaging the values of a certain number of preceding periods. Raw climate data is often noisy, and identifying meaningful patterns or trends requires smoothing the data.

- Data Grouping:

Climate data is captured every four hours. To reduce the data, we propose utilizing the average of all the records taken on a particular day, therefore producing a single daily value.

Statistical Metrics

- **Pearson coefficient:**

The Pearson correlation coefficient quantifies the linear relationship between two variables. It also asseses the degree of correlation between different meteorological parameters (e.g., temperature, humidity).

Employed within the correlation matrix to identify and isolate uncorrelated features with the target variable.

Pearson Correlation Formula

THE MATH EXPERT

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{[\, n\Sigma x^2 - (\Sigma x)^2 \,]\,[\, n\Sigma y^2 - (\Sigma y)^2]}$$

Statistical Metrics

- Z-score:

the Z-score, which measures how many standard deviations a data point is from the mean of a distribution. By calculating the Z-score for each data point and comparing it to a predefined threshold,outliers can be identified and filtered out.

Z Score Formula

$$Z = \frac{X - \mu}{\sigma}$$

University of Luxembourg

Technical Deliverable

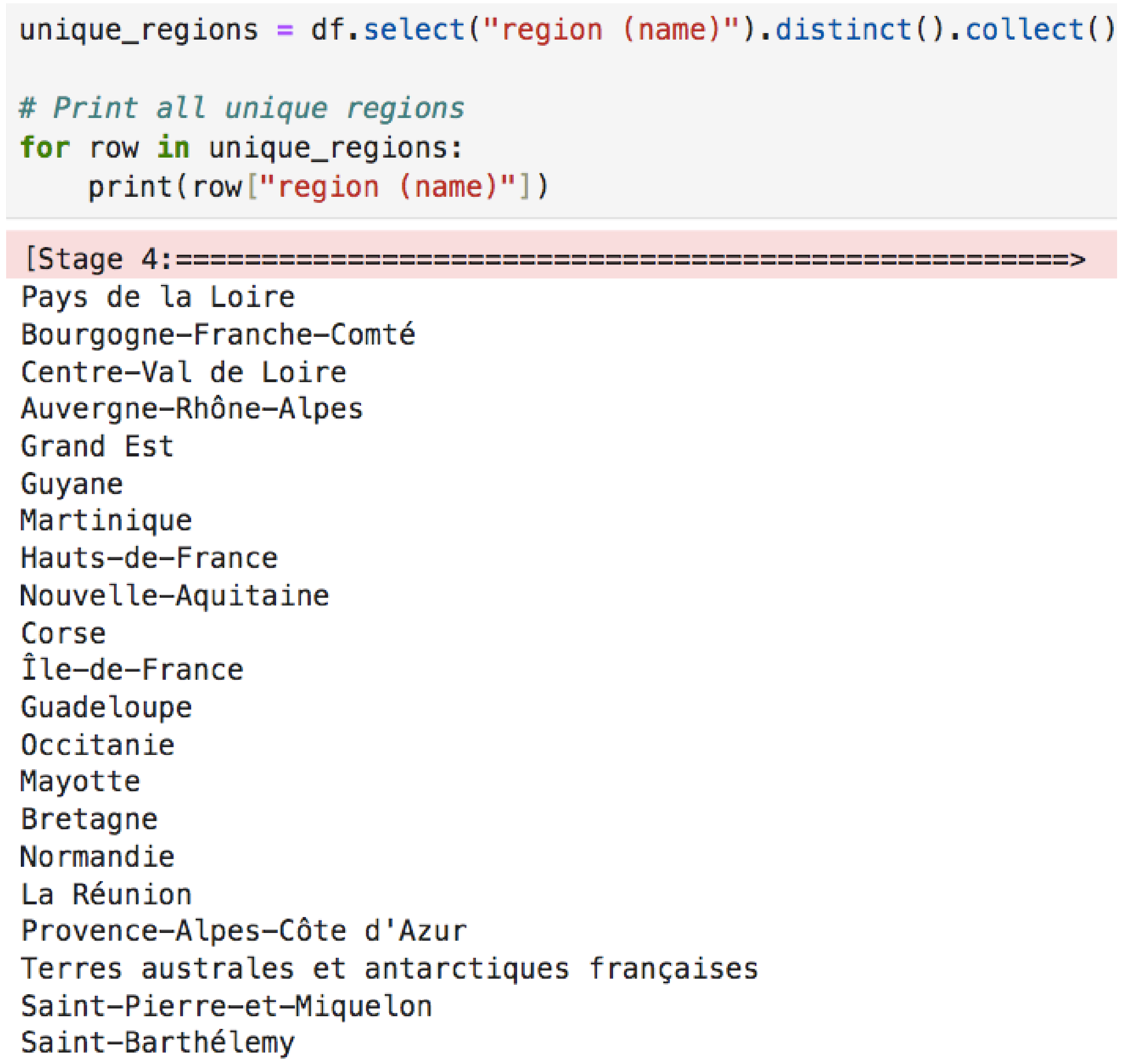
Explanation of the design and production of the technical part.

Exploratory Data Analysis

1. **Data overview:**

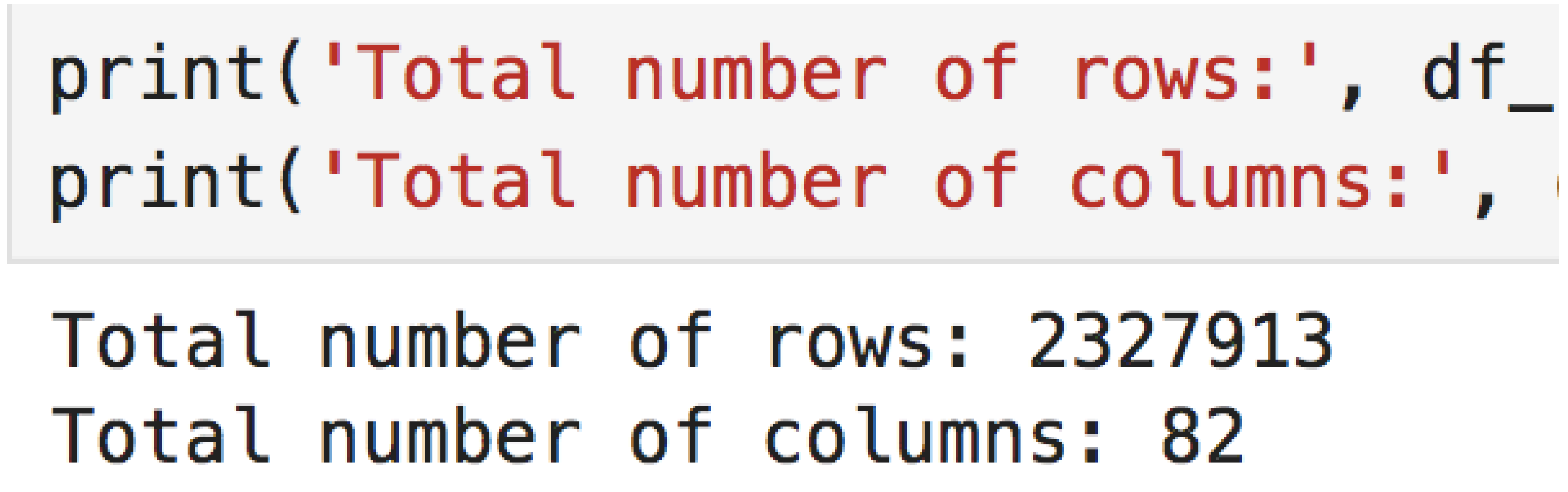
   - The dataset starts with 82 columns.
   - All columns are of string datatype.
   - All features are set to contain null values.
   - Total number of rows is 2327913.
   - Total of 62 weather stations contributed to this work.
   - 21 French regions are studied, distributed across different continents.
   - Records were collected from 01/2010 to 07/2023.

Exploratory Data Analysis



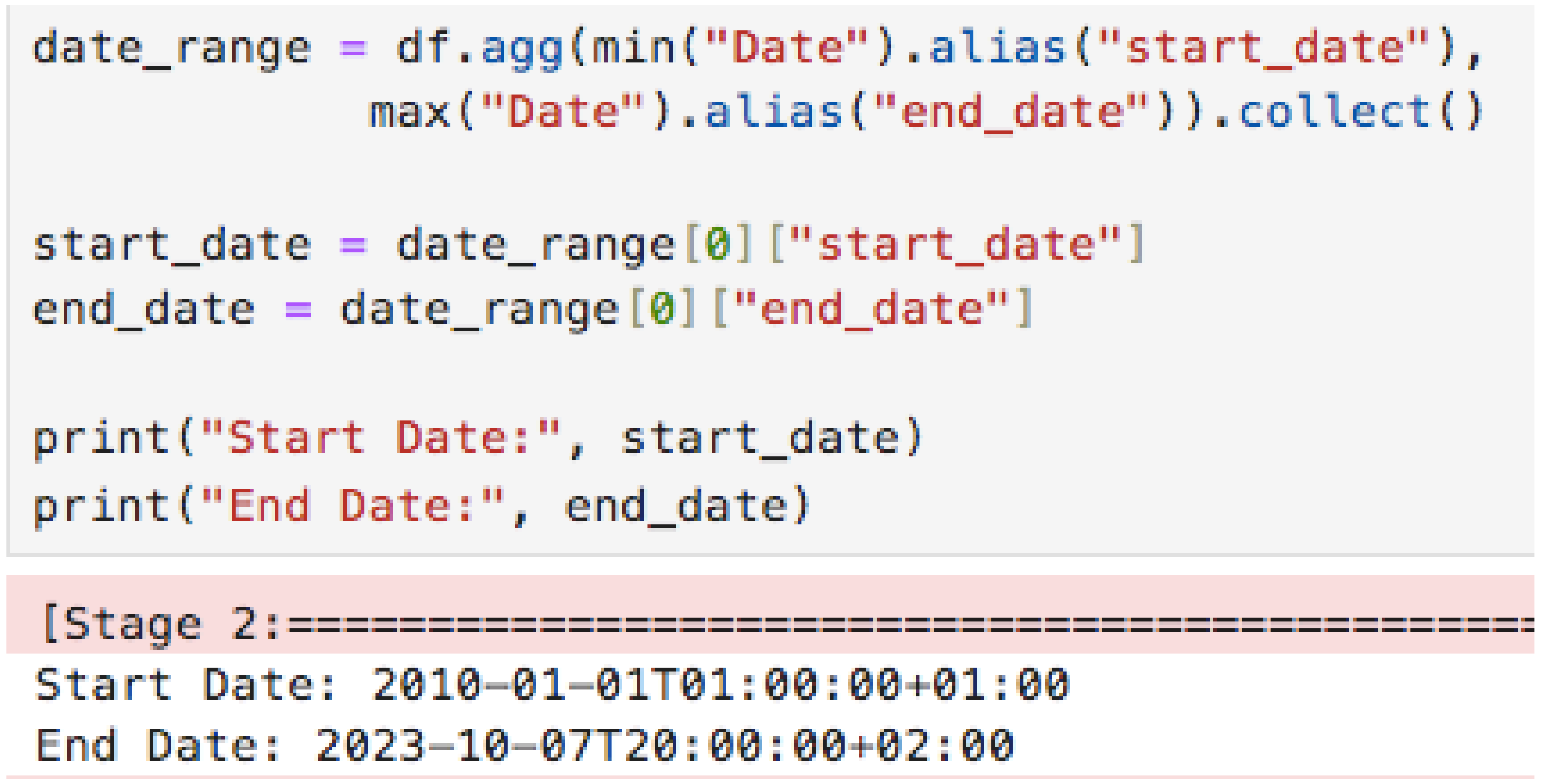
```
unique_regions = df.select("region (name)").distinct().collect()

# Print all unique regions
for row in unique_regions:
    print(row["region (name)"])
```

```
[Stage 4:=====================================================>
Pays de la Loire
Bourgogne-Franche-Comté
Centre-Val de Loire
Auvergne-Rhône-Alpes
Grand Est
Guyane
Martinique
Hauts-de-France
Nouvelle-Aquitaine
Corse
Île-de-France
Guadeloupe
Occitanie
Mayotte
Bretagne
Normandie
La Réunion
Provence-Alpes-Côte d'Azur
Terres australes et antarctiques françaises
Saint-Pierre-et-Miquelon
Saint-Barthélemy
```



```
print('Total number of rows:', df_
print('Total number of columns:',
```

```
Total number of rows: 2327913
Total number of columns: 82
```



```
date_range = df.agg(min("Date").alias("start_date"),
                    max("Date").alias("end_date")).collect()

start_date = date_range[0]["start_date"]
end_date = date_range[0]["end_date"]

print("Start Date:", start_date)
print("End Date:", end_date)
```

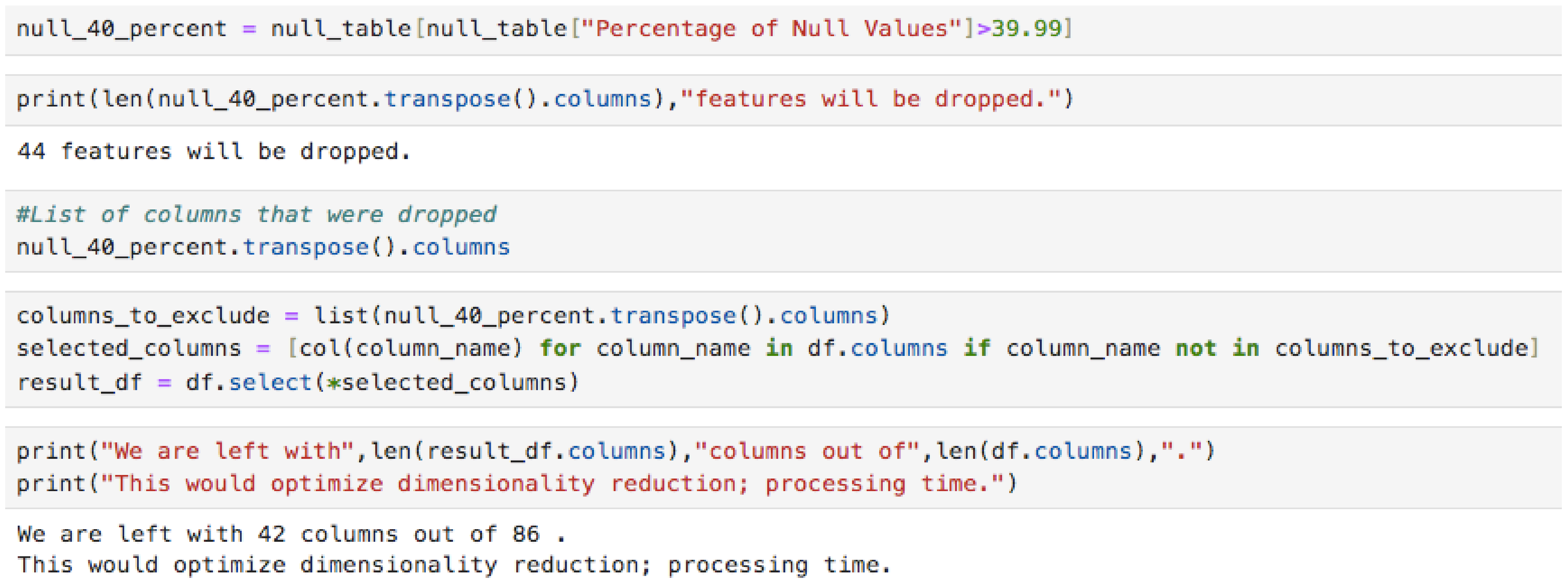
```
[Stage 2:==============================================
Start Date: 2010-01-01T01:00:00+01:00
End Date: 2023-10-07T20:00:00+02:00
```

Exploratory Data Analysis

2. Features Engineering

- <u>Missing values handling:</u>

  Some columns contain more than 95% missing values. A threshold is set to drop columns with a specific percentage of missing values.



```
null_40_percent = null_table[null_table["Percentage of Null Values"]>39.99]
```

```
print(len(null_40_percent.transpose().columns),"features will be dropped.")
```

```
44 features will be dropped.
```

```
#List of columns that were dropped
null_40_percent.transpose().columns
```

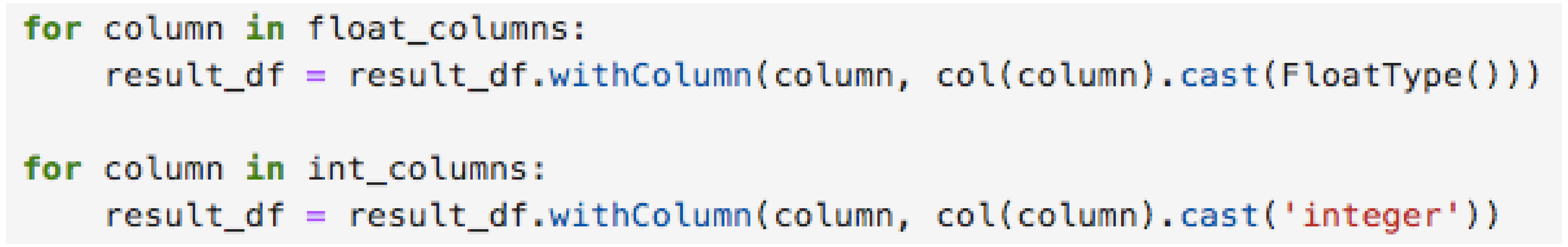
```
columns_to_exclude = list(null_40_percent.transpose().columns)
selected_columns = [col(column_name) for column_name in df.columns if column_name not in columns_to_exclude]
result_df = df.select(*selected_columns)
```

```
print("We are left with",len(result_df.columns),"columns out of",len(df.columns),".")
print("This would optimize dimensionality reduction; processing time.")
```

```
We are left with 42 columns out of 86 .
This would optimize dimensionality reduction; processing time.
```

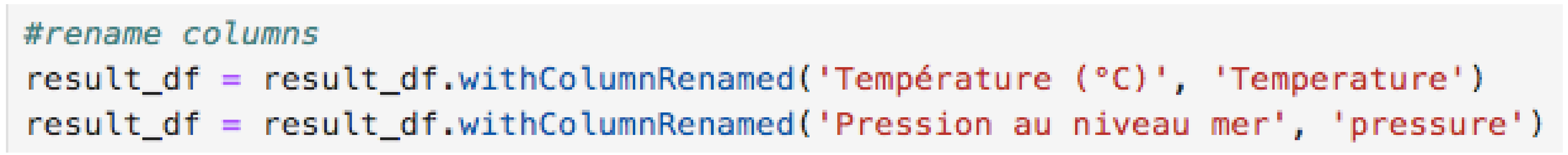
Exploratory Data Analysis

- Converted numeric columns to integer format.
- Renamed features with concise English names for clarity.
- Manually selected only the relevant columns for our weather analysis project.
- Impute with the mean value for the month where the missing data entry occurs.



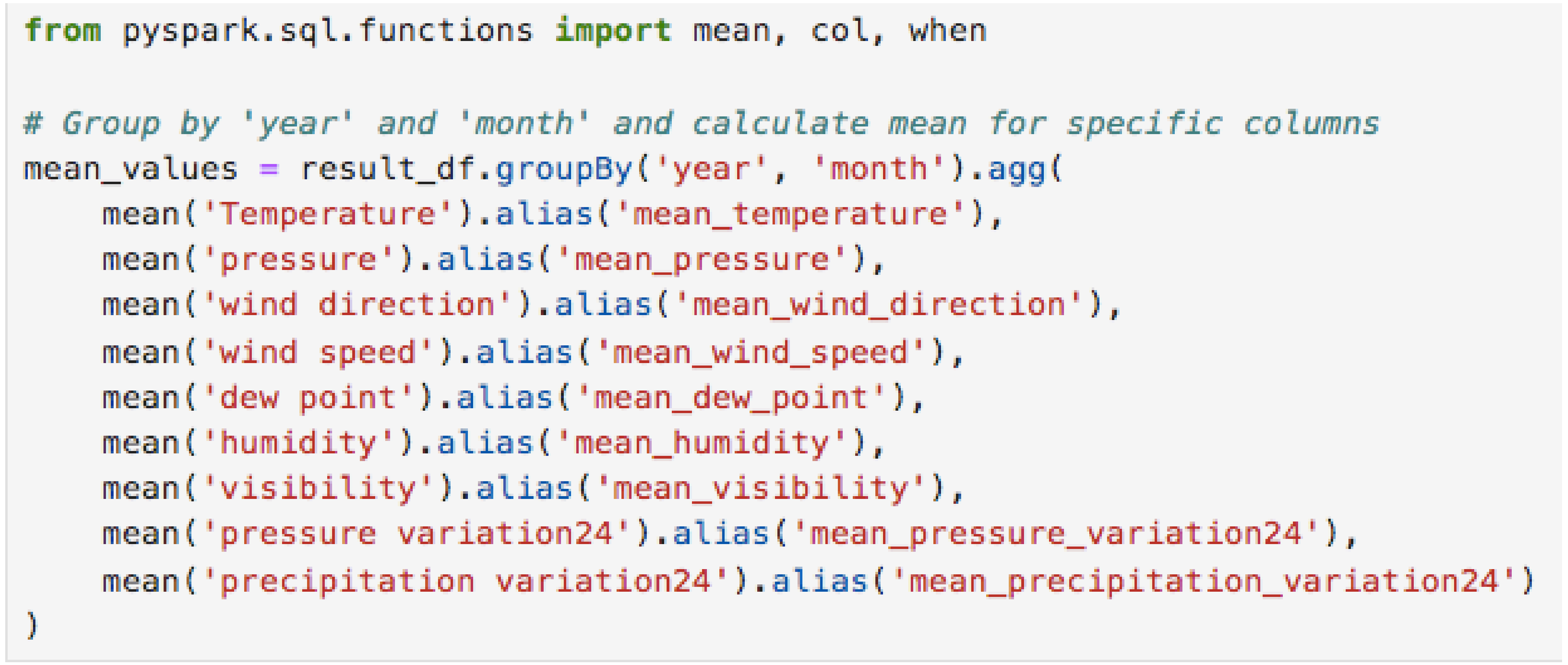
```
for column in float_columns:
    result_df = result_df.withColumn(column, col(column).cast(FloatType()))

for column in int_columns:
    result_df = result_df.withColumn(column, col(column).cast('integer'))
```

- Type conversion



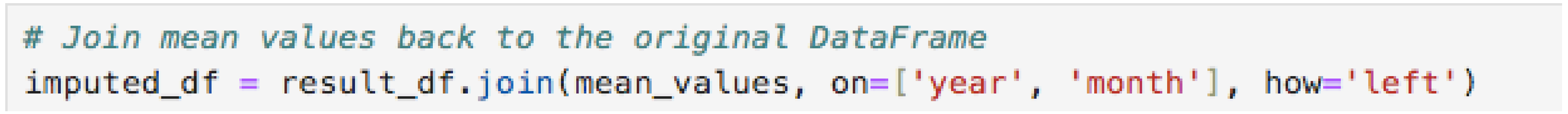
```
#rename columns
result_df = result_df.withColumnRenamed('Température (°C)', 'Temperature')
result_df = result_df.withColumnRenamed('Pression au niveau mer', 'pressure')
```

- Renaming columns



```
from pyspark.sql.functions import mean, col, when

# Group by 'year' and 'month' and calculate mean for specific columns
mean_values = result_df.groupBy('year', 'month').agg(
    mean('Temperature').alias('mean_temperature'),
    mean('pressure').alias('mean_pressure'),
    mean('wind direction').alias('mean_wind_direction'),
    mean('wind speed').alias('mean_wind_speed'),
    mean('dew point').alias('mean_dew_point'),
    mean('humidity').alias('mean_humidity'),
    mean('visibility').alias('mean_visibility'),
    mean('pressure variation24').alias('mean_pressure_variation24'),
    mean('precipitation variation24').alias('mean_precipitation_variation24')
)
```

- Mean value imputation



```
# Join mean values back to the original DataFrame
imputed_df = result_df.join(mean_values, on=['year', 'month'], how='left')
```

- Left join

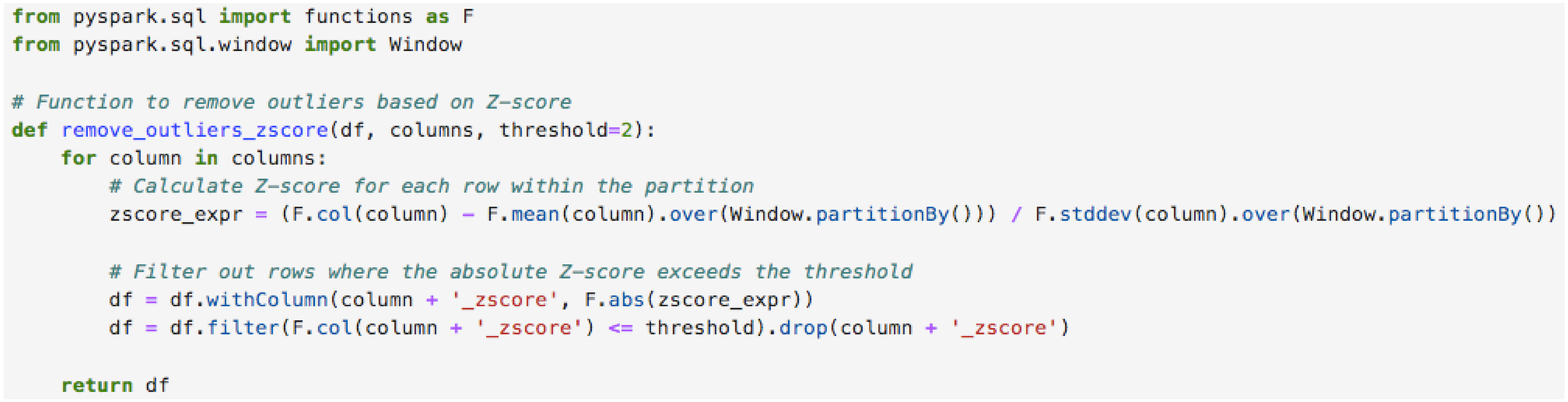
Time Series Analysis

- Outliers removal:

Visually evaluate the data skewness, to decide on Z-score threshold.

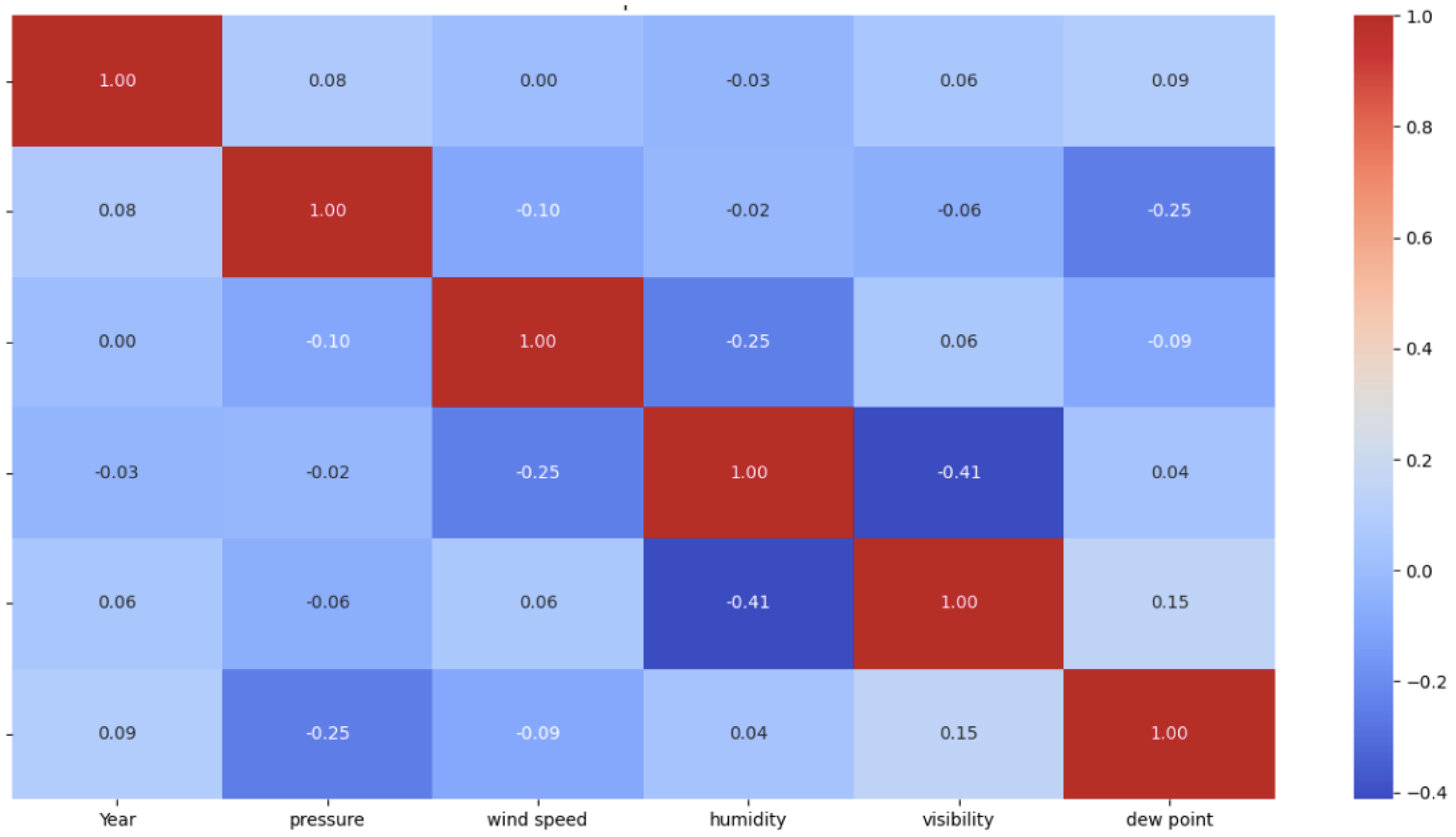
Applied Z score to identify and quantify outliers in the dataset.

Removing outliers ensures robustness and reliability in subsequent analysis.



```python
from pyspark.sql import functions as F
from pyspark.sql.window import Window

# Function to remove outliers based on Z-score
def remove_outliers_zscore(df, columns, threshold=2):
    for column in columns:
        # Calculate Z-score for each row within the partition
        zscore_expr = (F.col(column) - F.mean(column).over(Window.partitionBy())) / F.stddev(column).over(Window.partitionBy())

        # Filter out rows where the absolute Z-score exceeds the threshold
        df = df.withColumn(column + '_zscore', F.abs(zscore_expr))
        df = df.filter(F.col(column + '_zscore') <= threshold).drop(column + '_zscore')

    return df
```

Time Series Analysis

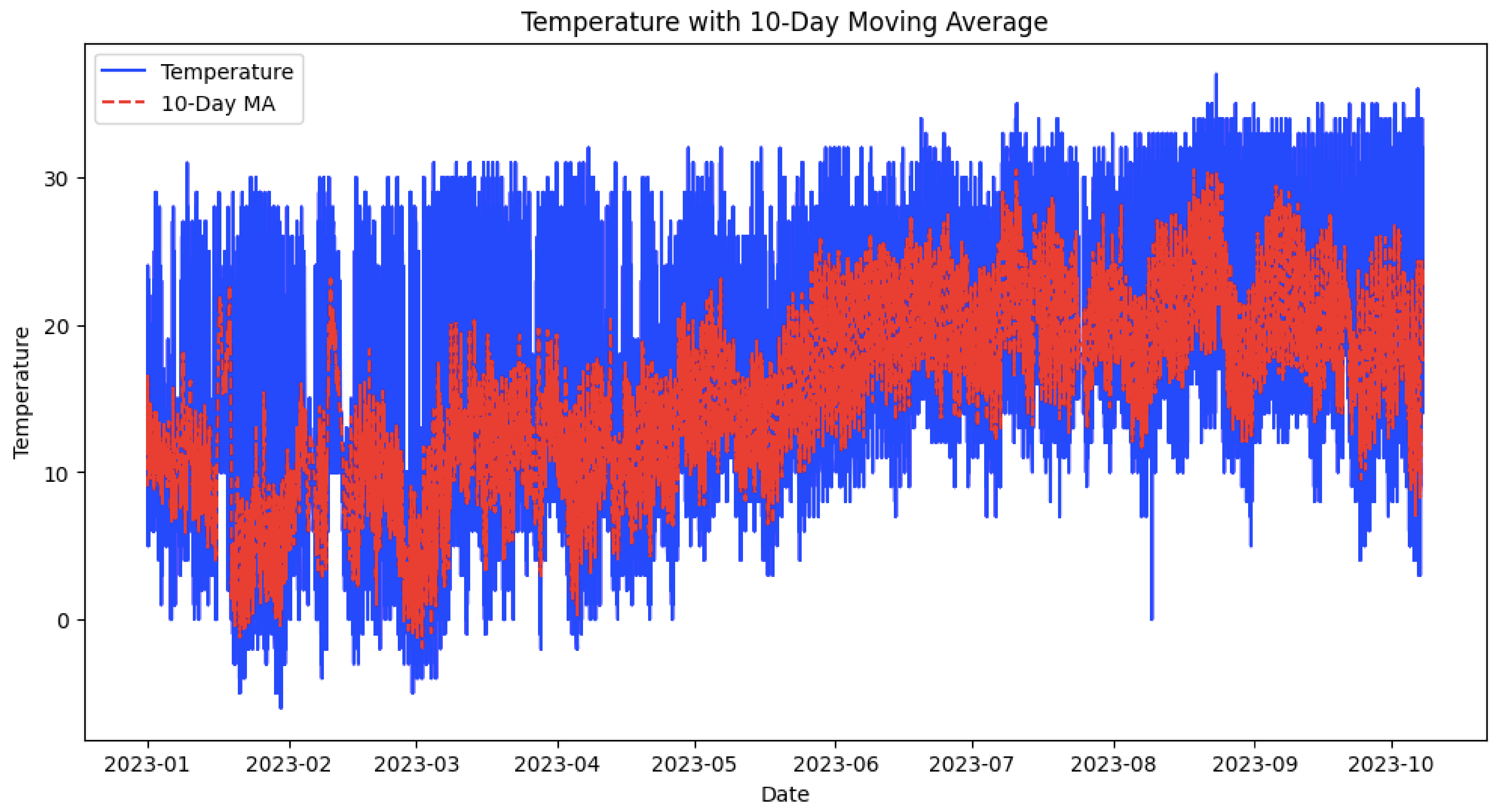


Year	pressure	wind speed	humidity	visibility	dew point
1.00	0.08	0.00	-0.03	0.06	0.09
0.08	1.00	-0.10	-0.02	-0.06	-0.25
0.00	-0.10	1.00	-0.25	0.06	-0.09
-0.03	-0.02	-0.25	1.00	-0.41	0.04
0.06	-0.06	0.06	-0.41	1.00	0.15
0.09	-0.25	-0.09	0.04	0.15	1.00

Correlation Matrix

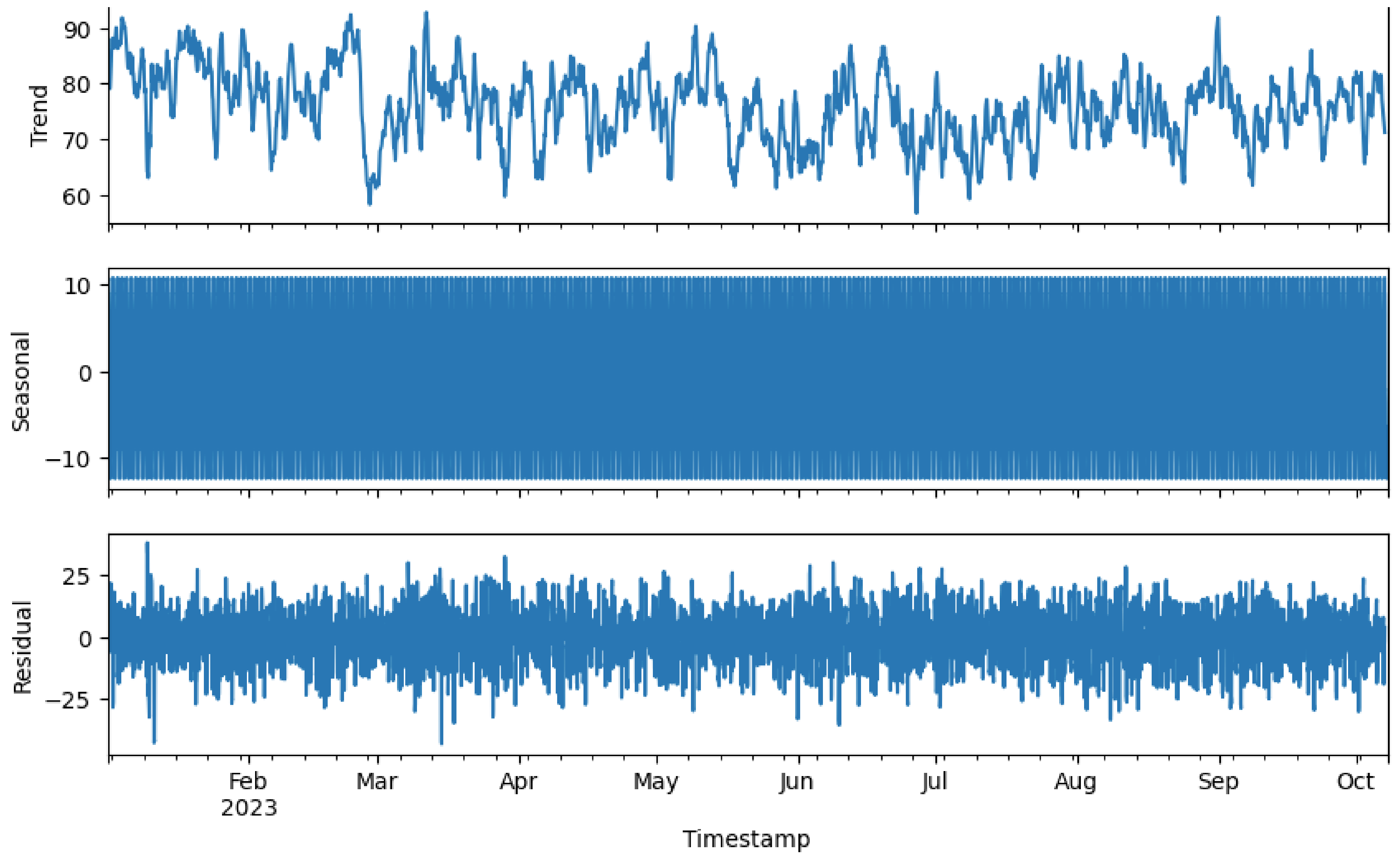
Time Series Analysis

- Moving Average



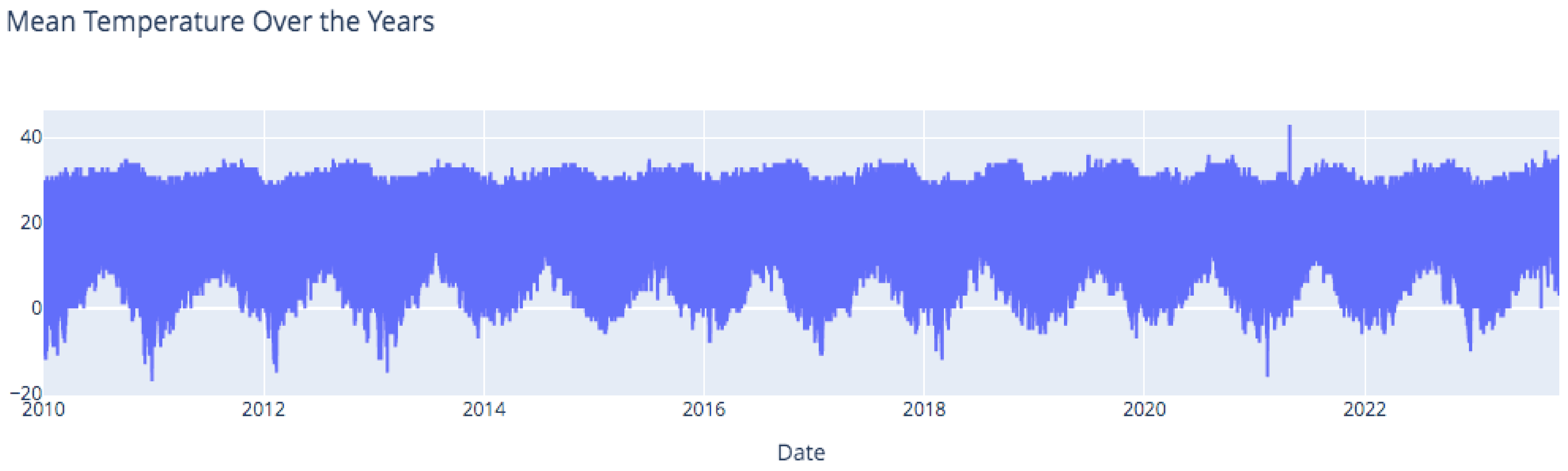
Time Series Analysis

- Seasonal Decomposition

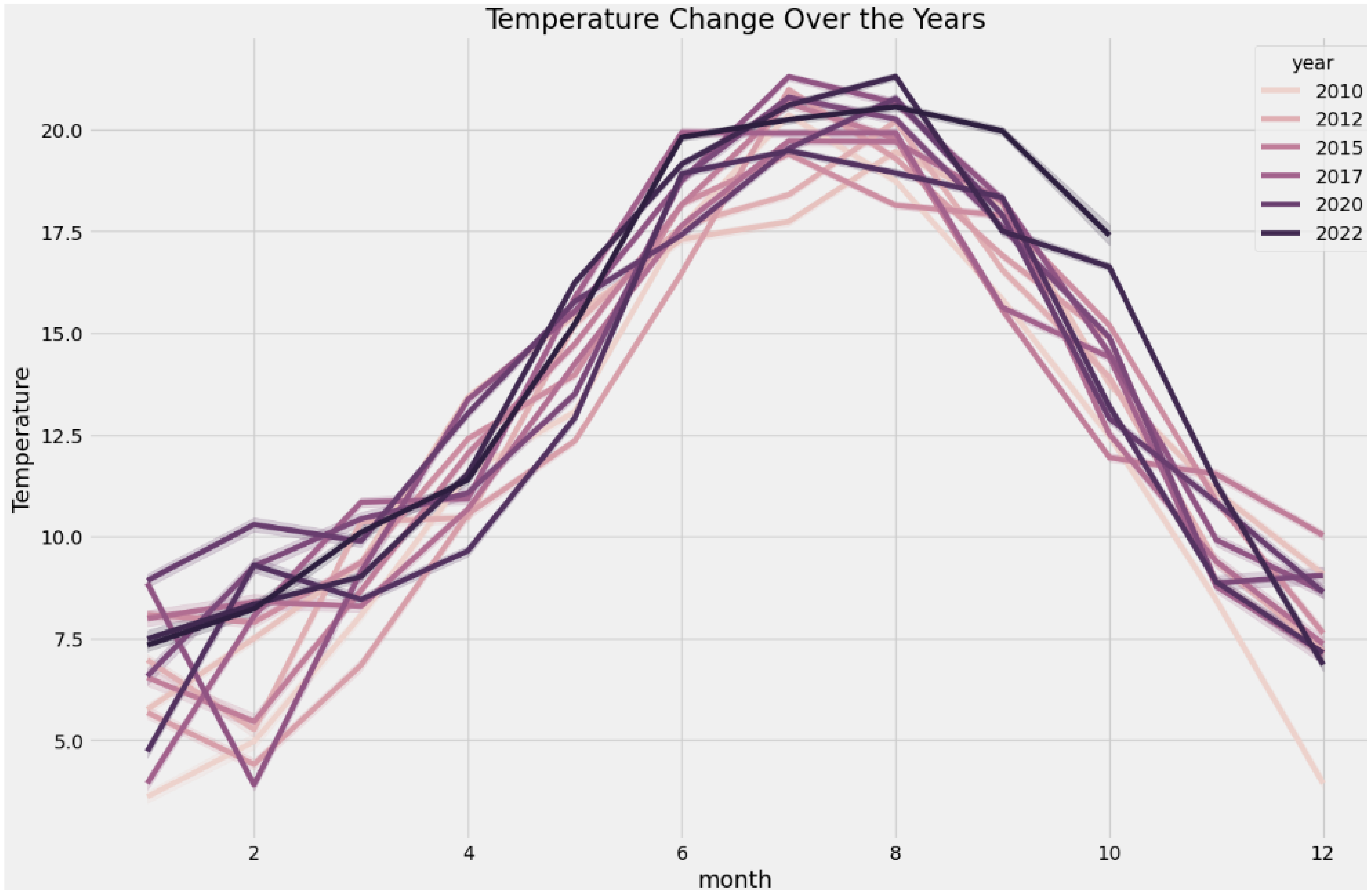


Time Series Forecsting

There appears to be a recurring pattern every two years - a seasonality trend.



Time Series Forecsting



Prophet Model

- Model description

Forecasting Results

- Dashboard

Dashboard

- Dash

Demo

- Dash