Climate Insights: Time Series Forecasting with Apache Spark

Sunday 14th January, 2024 - 15:45

Bacem ETTEIB
University of Luxembourg
Email: bacem.etteib.001@student.uni.lu

This report has been produced under the supervision of:
Giacomo DI TOLLO
University of Luxembourg
Email: giacomo.ditollo@ext.uni.lu

Abstract

In this document, we outline the comprehensive overview of Climate Insights, a project developed by Bacem ETTEIB, under the supervision of Prof. DI TOLLO. The project highlights distributed processing systems capabilities (Apache Spark) for real-time analysis performed on meteorological data, aiming to thoroughly understand and predict atmospheric phenomena. This work employs PySpark, a Python API for Apache Spark, Spark Streaming and cloud computing (Google Cloud) to process large meteorological data in real-time. The analysis, performed in a mutli-cluster environment, delivers comprehensive and useful insights accessible via a user-friendly web interface powered by Streamlit. This abstract highlights the project's importance and its potential impact on improving weather forecast techniques while relying on distributed processing systems.

1. Introduction

How can we combine the distributed processing system Spark with advanced time-series data analysis to improve the accuracy of weather forecasting applications?

In recent years, the combined efforts of scientists and the flourishing of the data science community allowed for better access and development of cutting-edges technologies designed for faster and proper data manipulation. As a result, the number of open-source projects directed towards analyzing and overcoming long-existing challenges has experienced a rapid surge. Specifically, meteorological patterns analysis and weather forecasting have long been critical challenges due to the large-scale and complex data related to the field of meteorology. The ability to accurately handle vast amounts of data and establish statistical relations among a high-dimension set of available features holds the key to enhancing our understanding of underlying climate dynamics. In response to these challenges, and under the expert guidance of Prof. DI TOLLO, we present Climate Insights, a project that lies at the intersection of advanced technologies and meteorological science.

Climate Insights is built on top of a sophisticated architecture leveraging cutting-edge technologies. Apache Spark, a key component of the project, is a powerful distributed processing system[1] enabling parallel processing of large-scale meteorological data. Unlike the traditional disk-based approach, Spark leverages its computational abilities by utilizing an in-memory processing approach. Therefore, Spark offers a different and more efficient memory management system which highlights one of the main selling points of the framework. Furthermore, cloud computing, specifically Google Cloud, offers the possibility to deploy and run projects in a multi-cluster environment, simulating Spark's behavior across different nodes.

The ultimate goal of Climate insights is not merely the implementation of statistical approaches for data analysis but the transformation of massive raw data into actionable insights through the introduced pipeline. For this reason, we present another key component of the project, Streamlit, allowing for a seamless integration of the produced results on an interactive web interface, updated continuously to ensure almost a 'real-time' connection with the data source.

Through this report, we provide an in-depth explanation of the scientific and technical aspects behind the development of Climate Insights. In the next sections, we delve into the design and the production of the proposed solution including the main algorithms, techniques and approaches involved in the scientific part. Simultaneously, the technical sections will dive deeply into the data processing workflows and the general pipeline that make the core of Climate Insights. Our aim is to thoroughly explain the potential of distributed processing systems combined with cloud computing in weather analysis applications while also providing a demonstration of the technical implementation of the aforementioned technologies.

2. Project Description

Climate Insights is conceived with the primary goal of assisting meteorologists and weather enthusiasts with the necessary tools to manipulate and analyze extensive meteorological datasets without having to worry too much about the data analysis phases. The communication between the application and the researcher is based on a user-friendly interface designed for seamless manipulation of an hourly updated flow of data, sourced from OpenDataSoft[8]. The dataset compromises a large number of recordings collected from 62 weather stations distributed across 21 French departments, including overseas regions. We use a set of technical tools to handle the large-scale data properly and to ensure a controlled usage of power without exhausting the available computational resources.

As mentioned previously, the computational exigencies associated with loading and processing a given dataset are contingent upon its size. Therefore, we conclude that a significant increase in the input data size necessitates augmented computational power. Climate Insights tackles the computational demands posed by large-scale meteorological data through the integration of Apache Spark's distributed processing capabilities. The main idea behind the implementation of Apache Spark is its efficiency in leveraging computational power and minimizing costs through the concept of parallel processing. In short, the parallel processing of data allows the concurrent execution of tasks on distributed data among multiple nodes (worker nodes)[1]. In addition, Spark's in-memory processing approach facilitates parallel processing, ensuring efficient management of substantial datasets. Given the constraints of data size and limited resources in the context of our project Climate Insights, the implementation of Apache Spark aligns seamlessly with our main objectives of effectively handling large-scale data. To enhance scalability and accessibility, Climate Insights is deployed on Google Cloud, enabling multi-cluster simulation of Spark's behavior. The choice of Google Cloud was based on multiple criteria. Firstly, Google offers a credit of 400 dollars to explore freely the available tools and get a grasp of the available functionalities. This offer allows for the usage of Google Cloud's resources, mainly the distributed processing frameworks, in a cost effective way. Secondly, the choice was also strongly driven by the solid and robust infrastructure of the cloud. Those criterias underscore the project's commitment to scalability and accessibility.

The project serves as a successful manifestation of the integration of complex data processing techniques with the framework of a distributed processing system, namely Spark. The transformation of the raw data collected originally from OpenDataSoft into insightful results and comprehensive visualizations involves rigorous data cleaning procedures, handling anomalies, missing values and additional features engineering techniques. In the technical section of this report, we dive more into the details of those procedures and we explore the different functions deployed for the task of data manipulation.[9]

3. Main required competencies

The scientific and technical competencies covered in this section represent the knowledge possessed by the author and that is critical for the understanding of the project's structure and development. Thus, they define the pre-condition for this project. In summary, the implementation of Spark on top of Google Cloud and time-series data analysis represent new technical concepts for the author. Therefore, we dive into the scientific and technical backgrounds of the mentioned approaches.

3.1. Scientific main required competencies

The scientific competencies required for the development of Climate Insights include a basic understanding of meteorology and atmospheric sciences. Given that our data frame contains climate features, it is of utmost importance to understand the different meanings and interpretations of weather features such as wind direction and dew point. In addition, as we are performing data analysis and driving interpretations from observed patterns, it's crucial to understand the impact of different meteorological factors on weather conditions. Besides, knowledge of statistical methods and data analysis techniques is fundamental. This includes the understanding of regression analysis and time series analysis.

3.2. Technical main required competencies

The technical competencies necessary for this project include expertise in Python and libraries used for data analysis such as Pandas and Matplotlib. Although we will be mainly using PySpark, an understanding of Pandas data frames and their structure is fundamental. Moreover, proficiency in working with large datasets and data analysis techniques such as data cleaning, feature engineering and transformation is of utmost importance.

Besides, as Google Cloud platform will be used for Spark distributed processing simulation, a basic knowledge of cloud computing platforms is required. The cloud service offers the possibility to run code on multiple clusters and scale the project to handle large datasets efficiently. Finally, we require experience using web interface tools like Streamlit intended to create interactive and user-friendly interfaces.

4. Scientific Deliverable: Distributed Processing Systems

The scientific question to be answered through this report is:

How can we combine the distributed processing system Spark with advanced data analysis to improve the real-time accuracy of weather forecasting?

Advancements and progress in the fields of meteorological research and weather forecasting applications have hugely profited from the flourishing of the data science and distributed computing domains. As more attention is being drawn to exploratory data analysis and cloud computing, researchers are constantly improving the cloud infrastructure and the data analysis pipelines. In contrast to traditional weather analysis methods that often struggle with large and complex datasets, modern data engineering techniques and especially data processing technologies like Spark have become essential in overcoming these challenges. Through this project, we aim to dive into the scientific aspects of data processing and the application of distributed processing systems, mainly Spark, to improve large and complex data analysis accuracy. We focus on using Apache Spark Streaming for real-time analysis and we rely on meteorological data related to multiple french regions.

1) **Context:**

The primary goal of this project is to leverage Spark's Streaming performance on large and complex datasets. Through the study of meteorological data, which includes the analysis of atmosphere and weather patterns, our aim is to understand and forecast atmospheric phenomena in real-time. Computer science, particularly, data analysis, provides the necessary techniques to load, process and analyse the different patterns present in the data. Therefore, these analyses will be handy for a weather specialist to drive useful insights. The study of the book "Advanced Analytics with Spark 2nd edition" [1] allowed us to thoroughly understand the power of distributed processing systems in our context. The various examples provided by the author, in addition to the detailed explanation of Spark and Scala based applications allowed for a deep understanding of the process of developing data applications based on Spark. Thus, the motivation behind our scientific question is to leverage the power of Apache Spark, essentially PySpark, and cloud computing to address the challenges presented by the large size of data sets and the limited computational power, aiming for more accurate weather predictions.

2) **Sub-questions:**

a) ***How can we implement advanced data analysis and modelling techniques to enhance the identification and prediction of weather patterns?***

We explore different articles and research papers to highlight the key techniques of data preprocessing and manipulation, namely the survey on data stream frameworks, analysis and algorithms [3]. As explained in the included reference, big data provides a valuable knowledge and plays a substantial role in today's services and infrastructures. To gain the maximum benefit from the huge amount of data and to unleash its potential, we need to apply "real-time analysis, predictions and forecasts" [3]. Therefore, it is of utmost importance to create a pipeline that is able to process big data in real time, apply different data analysis techniques such as formatting, cleaning and creating visualizations. We commonly refer to this process as data engineering. In fact, it is considered to be the most important and time consuming task for data scientists. Afterwards, we examine the modelling phase, where we apply forecasting or prediction methods using the appropriate model on the newly generated data frame. It is also worth mentioning that we use distributed systems, mainly Apache Spark, to handle the large data through the multi-clusters environment. Unlike traditional methods, or more precisely single machine data processing, the distributed system architecture enables parallel processing across multiple nodes, allowing for a seamless scalability[4].

b) ***Given the large size and the complex format of the data, what pre-processing techniques are the most effective for accurate analysis?***

Thanks to the great work provided by the German Climate Service Center [6] to illustrate the different statistical methods for the analysis of simulated and observed climate data, we explore the presence of weather patterns across 62 regions in France, over a span of 10 years. We process the data accordingly and create pipelines to apply those methods in the newly fetched data, as per user request. Below, we highlight some of the key methods depicted by the German Center.

- **General statistical methods**
  We explore the Pearson's product moment correlation coefficient to study the relationships between the existing variables. The reason why to choose Pearson's correlation over other measures is that this metric is useful when the data exhibits non-linear relationships. Another reason is the presence of outliers, as Person's coefficient is robust to those issues.
- **Frequency distributions**
  When used on time series data with even spacing, and applied on variables like temperature and precipitations, the descriptive statistics method "Relative frequency distribution" can be highly useful when it comes to anomaly detection, pattern recog-

nition and comparative analysis.

- **Time series analysis**
  Time series analysis plays a major role in driving meaningful and insightful patterns and trends within the weather data. We take, as an example, the ”Running mean and Running median” measures. By calculating the average of data points, or finding the middle value within a moving window, we can obtain a smoothed representation of the series and spot outliers.

c) ***How can distributed processing with PySpark enhance the speed and efficiency of weather forecasting application?***

Based on the book ”Real-Time Big Data Analytics” [5], we explain the theoretical foundations of distributed processing systems and their ability to handle large-scale weather data. The book explains the architecture of Spark, the Resilient Distributed Datasets (RDDs) and how they scale computations horizontally, allowing to understand large-scale data without sacrificing computational power and most importantly computational speed.

3) **Ideas:**

a) Advanced Data Analysis Techniques:
Over the recent years, machine learning algorithms showed great performance in different scientific fields, including meteorology. Algorithms like Decision Trees and Support Vector Machines (SVM), proved their ability to recognize patterns in meteorological data that are hard to spot with traditional methods. Additionally, ensemble methods, like random forests and gradient boosting, have a great performance in predicting weather related phenomena thanks to their ability to combine the strengths of multiple algorithms.
b) Spark Streaming:
We have witnessed so far the capabilities of Spark in managing large-scale data consumption through parallel processing. Yet, Apache Spark is not only limited to the processing of data for batch use cases. It also extends its abilities to offer the possibility of processing data in real-time from various distributed data streams with low latency.
c) Distributed Processing with PySpark:
The use of frameworks such as PySpark to enable distributed processing has revolutionized the field of big data analysis. PySpark divides the computing load among several nodes and speeds up processing considerably by enabling parallel processing. Weather forecasting models may run simulations, convert data, and run sophisticated algorithms simultaneously by utilizing PySpark’s features. This distributed technique allows for the scalability needed to manage massive volumes of meteorological data, while simultaneously improving analysis speed. Through the simulation of cluster behavior on cloud computing platforms such as Google Cloud, one can gain insights into the scalability of the system and optimize cluster configurations.

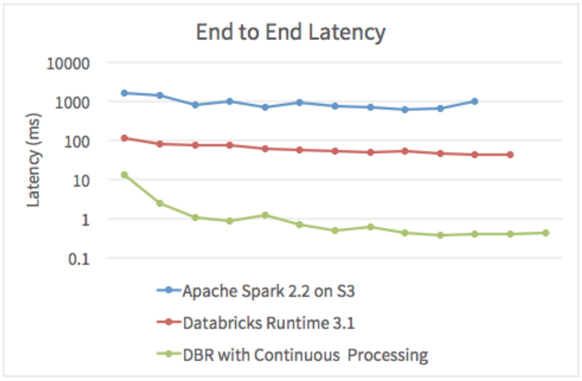


Fig. 1: image imported from Databricks

5. Design

Time series analysis stands as a pivotal element in contemporary applications across diverse domains, playing a fundamental role in shaping decision-making processes. Whether applied in the financial sector to forecast market trends or employed in weather forecasting applications to anticipate complex climate patterns, the extraction of insightful information from time-stamped data is imperative[10]. The temporal dimension inherent in time series data facilitates the discernment of patterns, trends, and anomalies, thereby empowering strategic decision-making.

Moreover, utilizing the temporal feature as a default index for data introduces a remarkable level of flexibility, allowing for the resampling of data points into various frequency distributions. This flexibility extends from an hourly to a yearly grouping, enabling a nuanced exploration of temporal trends and patterns. However, it is crucial to acknowledge that the inclusion of timestamps contributes to exponential data growth, given the continuous nature of time, which can be subdivided in various ways. For instance, the weather stations continuously collect data every 4 hours. This exponential increase in data size raises pertinent concerns, particularly when deploying systems constrained by limited resources, such as disk space and memory.

The investigation of time series data, driven by the imperative to address challenges arising from data size, compels us to explore technologies explicitly designed to handle substantial volumes of data efficiently. This exploration prompts a critical examination of the transition from a single-core system to operating within a distributed processing environment. In this paradigm, the task of data analysis is distributed among multiple agents, specifically worker nodes, as elucidated in the seminal work conducted by Matei Zaharia et al.[4].

The foundational work by Zaharia underscores the merits of distributed processing environments, exemplified by frameworks like Apache Spark. These environments offer a scalable solution to the challenges posed by extensive time series datasets, allowing for parallelized computation and analysis across multiple nodes. The utilization of distributed processing becomes particularly pertinent when confronted with the need to manage and manipulate data at a scale that surpasses the capacity of a single computational unit.

This shift towards distributed processing aligns with the goals of projects like Climate Insights, which necessitate the effective handling of vast and intricate meteorological datasets. The distributed processing paradigm, exemplified by the capabilities of Apache Spark, not only addresses concerns related to data size but also harnesses the computational power of multiple nodes to enhance efficiency in time series analysis[5]. The subsequent sections of this discourse will delve deeper into the nuanced considerations and design choices that underpin the utilization of distributed processing systems, specifically within the context of Climate Insights.

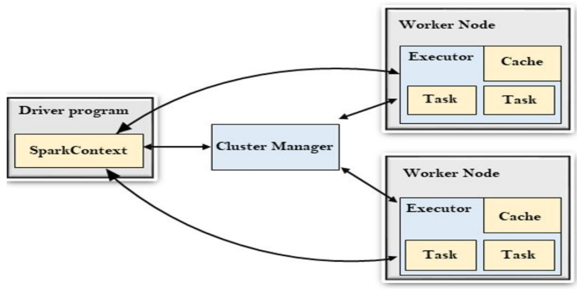
6. Production

The implementation of the scientific aspect of Climate Insights involves a carefully selected technology stack. The core technologies include Apache Spark for distributed processing, Google Cloud for scalable infrastructure, and Dash for real-time visualization. The aforementioned concepts in the context of time series analysis highlight the core of our scientific production. We follow a top-bottom approach to illustrate the adapted methodology and the undertaken steps to produce the conceptual design of Climate Insights.

6.1. Distributed Processing Systems

Distributed processing systems offer scalable solutions to meet the ever-expanding demands of data-intensive applications. These systems operate by distributing computational tasks across a network of interconnected nodes, thereby optimizing resource utilization, enhancing performance, and mitigating bottlenecks associated with traditional, centralized processing. The strategic distribution of workload not only facilitates the parallel execution of tasks but also fosters fault tolerance, ensuring continued operation even in the face of node failures. In this dynamic context, the modeling and analysis of probability distributions become paramount, providing insights into the reliability, performance, and efficiency of distributed systems.

Apache Spark, an open-source, general-purpose distributed computing framework that has gained widespread acclaim for its versatility and efficiency, represents a pertinent example of distributed processing systems. Apache Spark distinguishes itself through its ability to process data in-memory, reducing the need for extensive disk I/O and accelerating iterative algorithms. This framework encompasses a comprehensive set of libraries, including Spark SQL, Spark Streaming, MLlib, and GraphX, offering an integrated ecosystem to address diverse data processing requirements.



6.2. Architecture of Apache Spark

6.2.1. Spark Driver

. Its primary function is to execute the main method of the program, serving as the entry point for user code execution. Upon execution, the Spark Driver creates the Spark context or Spark session. This initiation process involves establishing the essential foundations for Spark's distributed processing, including the creation of Resilient Distributed Datasets (RDDs), which serve as fundamental abstractions in Spark's data processing paradigm. Two primary functions define the Spark Driver's role:

Converting User Program into Tasks: The driver translates the high-level operations specified in the user's code into executable tasks. These tasks are the fundamental units of work that will be distributed and executed across the worker nodes.

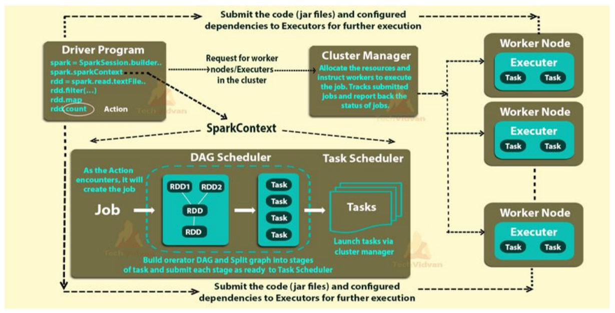
Scheduling Tasks on Executors: With the assistance of the chosen cluster manager (such as Standalone, Apache Mesos, or Hadoop YARN), the Spark Driver schedules the tasks for execution on the allocated executors. The cluster manager is responsible for launching and managing these executors dynamically, based on the application's resource needs and workload.

6.2.2. Spark Executor

. The Spark executors, initiated at the commencement of a Spark application upon job submission, persist throughout the entire lifespan of the application, forming a resilient foundation for continuous data processing. At their core, these executors are responsible for executing individual tasks within a Spark job, encompassing the actual programming logic for data processing, including transformations and actions on RDDs (Resilient Distributed Datasets) and data frames. Simultaneously, the executors diligently return results to the Spark driver, establishing bidirectional communication for the seamless flow of information. Beyond task execution and result delivery, Spark executors play an additional crucial role as repositories for in-memory storage, specifically catering to RDD datasets and data frames. This in-memory storage becomes particularly pivotal when users choose to cache or persist datasets, minimizing computational overhead associated with recalculating RDDs and enhancing the overall efficiency of Spark applications.

6.3. Apache Spark Workflow

The initial component in view is the driver program, activated upon a Spark submit. This program initiates a request for resources from the cluster manager while concurrently launching the main user processing program. Subsequently, the execution logic is processed, and the Spark context is established in parallel. The Spark context is utilized for various transformations and actions, leading to the accumulation of transformations in the Spark context as a Directed Acyclic Graph (DAG), forming the RDD lineage until an action is encountered. For instance, we can think of DAG in the context of Apache Spark as a set of tasks, represented in nodes while the edges connecting them illustrate the dependencies.


Submit the code (jar files) and configured dependencies to Executors for further execution
Driver Program
spark = SparkSession.builder..
spark.sparkContext
rdd = spark.read.textFile..
rdd.filter(...)
rdd.map
rdd.count
Action
Request for worker nodes/Executers in the cluster
Cluster Manager
Allocate the resources and instruct workers to execute the job. Tracks submitted jobs and report back the status of jobs.
Worker Node
Executer
Task
Task
Worker Node
Executer
Task
Task
SparkContext
DAG Scheduler
Task Scheduler
As the Action encounters, it will create the job
Job
RDD1
RDD2
RDD
RDD
Task
Task
Task
Task
Tasks
Launch tasks via cluster manager
Build orerator DAG and Split graph into stages of task and submit each stage as ready to Task Scheduler
Worker Node
Executer
Task
Task
Submit the code (jar files) and configured dependencies to Executors for further execution

Upon invoking an action, a job is created, constituting a collection of distinct task stages. These tasks are then dispatched by the cluster manager to the worker nodes with the assistance of the task scheduler. The conversion of RDD lineage into tasks is orchestrated by the DAG scheduler, which creates a DAG based on the program's diverse transformations. These transformations are divided into distinct task stages and presented to the task scheduler as tasks become ready.

Subsequently, these tasks are initiated on various executors within the worker nodes through the cluster manager. The comprehensive management of resource allocation, as well as the monitoring of jobs and tasks, is meticulously handled by the cluster manager. With every Spark submit, the user program and specified configurations are replicated across all available nodes in the cluster. This ensures that the program is locally accessible on all worker nodes, obviating the need for parallel executors on different nodes to engage in network routing.

6.4. Time Series Analysis

We have covered the essential techniques and frameworks to properly analyse large-scale data while maintaining an efficient usage of memory and computational power. In this part, we shift our focus from distributed processing systems to another important concept, namely time series analysis. As the name suggests, the data is distinguished by a temporal component that adds an additional layer of complexity and significance to the analysis. The temporal nature of climate data, with timestamps associated with each data point, enables the identification of patterns and anomalies that may not be apparent in static datasets. This temporal dimension allows for a more nuanced exploration of climatic changes, helping researchers and decision-makers comprehend the dynamic nature of environmental conditions. In this context, various statistical and machine learning techniques come into play. These include methods for trend analysis, seasonality detection, anomaly detection, and forecasting. The chosen approach depends on the specific goals of the analysis and the characteristics of the data. For instance, in Climate Insights, understanding long-term climate trends, detecting unusual weather events, and predicting future climate conditions are likely focal points. In our work, we focus on the forecasting of different metrics, chosen by the user input. For instance, one can choose to forecast temperature degrees over an upcoming period. Other possibilities will include pressure and humidity. In forecasting, historical data patterns are used to make predictions about future events. Based on the specifications of the project, the time range used to study those historical patterns can vary from granular hourly grouping to broader yearly grouping. It is of utmost importance to note that the results are prone to overfitting when considering a small time interval. For example, when forecasting temperature, using seconds as a time interval would not be efficient as we will be introducing more noise to the model.

7. Technical Deliverable: Weather Analysis Application

As we have explored the concept of distributed processing systems and Apache Spark framework, this section zooms in on the implementation of Spark on cloud computing platforms, especially Google Cloud. Our global objective is to smoothly leverage Spark's computational power through parallel processing by deploying it in a multi-cluster environment offered by Google Cloud services. It is worth mentioning that, through this project, we work on a large-scale dataset with a wide range of features. Therefore, it is of utmost importance to provide the user, more precisely the weather specialists, with an accessible, user-friendly and a real-time weather forecasting application. The intersection of Spark, Google Cloud and Streamlit, serving as the base and key frameworks for our application Climate Insights, promises to revolutionize the way weather data is processed, analyzed and delivered.

1) **Context:**

The study of distributed processing systems[1][3], especially Apache Spark, forms the baseline for our real-time weather forecasting application. We overcome the constraints of traditional computing infrastructures that rely on a single computing node by using Google Cloud's cloud distributed computing systems. When combined with Spark's prowess of parallel processing, we create a dynamic environment capable of managing real-time and large scale time series data[4]. This aggregation of the presented frameworks, empowers our application to smoothly distribute complex time consuming operations across multiple nodes in the Google Cloud's multi-cluster environment. Besides, it allows to create a pipeline for data processing, visualizations and modelling through a user-friendly and interactive web-interface, powered by Streamlit[7]. Our motivation behind this work is to revolutionize the landscape of weather forecasting. By providing weather specialists with an interactive and easily accessible tool to illustrate the different meteorological patterns presented in the fetched data, upon user request and up to the latest date provided by the data API query, we allow for faster data exploration and accurate modelling applications from forecasting to prediction of atmospheric phenomena.

2) **Functionalities:**

In this work, we present the summary of researches and experiments performed on meteorological data to come up with a robust and efficient system that is able to provide reliable and accurate analysis for the collected weather features. As we design our system, we focus on two main axis: computational power and processing time. Our aim is to minimize those factors while keeping a satisfying measure of accuracy. What is meant by accuracy is the assessed value of our predictions/forecasts in regards to what is expected as output. Hereby, we depict, in an abstract way, the main functionalities deployed as the core of our system.

- **Data Ingestion**
  During the design of Climate Insights, we give high priority to data collection and loading. As we are working with real-time weather data, we ensure a seamless flow of data into our application through the API provided by OpendataSoft. The query fetches and downloads data from the source while maintaining the right format, the frequency and the number of records. By keeping the system updated, we make sure that our front-end interface, containing different analysis plots and forecasts, is always updated with the most recent and relevant information.
- **Distributed Processing**
  The need to minimize computational power and to handle large-scale data properly is paramount in our weather analysis system. Therefore, we dive into the application of Apache Spark[4][5] through the referenced books, and its prowess to optimize computational resources effectively. Through its architecture based on distributing tasks across multiple nodes, we ensure the workload is split into multiple parts across the available machines, maximizing the overall computational power.
- **Google Cloud and Streamlit**
  To properly benefit from Spark's distributed processing power, we require a multi-cluster environment that allows the creation of a master node and multiple worker nodes. Google cloud offers the opportunity to simulate Spark's behavior on their machines. We use the cloud service, along with PySpark, a Python based API, to define the main tasks and split them over the nodes. In addition, we create a front-end, user-friendly and accessible interface for the weather specialists to access our work through the deployment of Streamlit[7].

3) **Ideas:**

Our application lies in the intersection of different cutting-edge technologies, namely Apache Spark and Google Cloud, establishing a robust and resilient framework. While Spark ensures data managing in a distributed processing system, Google Cloud offers a multi-cluster environment, a storage infrastructure and a high computing baseline, creating a tight bound between speed and scalability. The backend of our application relies heavily on the mentioned frameworks whereas for the front-end development, we focus solely on using Streamlit, as it provides the right tools to create a user-friendly interface while supporting PySpark and allowing a neat display of the different visualizations associated with our system.

In this technical deliverable, we focus on the seamless fusion between Apache Spark's parallel processing, Google Cloud's infrastructure and Streamlit custom web applications. This allows to revolutionize the analysis of meteorological data in real-time while also offering the ability to extend this deliverable to handle other features. Therefore, this work represents the entry point to shift the development of data science based projects into multi-cluster environments, allowing the efficient managing of large-scale data.

7.1. Design

Climate Insights adopts a distributed processing architecture with Apache Spark at its core. The choice of this framework stems from its ability to handle large-scale data efficiently. The architecture encompasses PySpark for Python integration, an API designed by the author for real-time data analysis, and Google Cloud for scalable cloud computing. In the initial part of this report, we highlighted the imperative for a multi-cluster environment to effectively manage the substantial volume of data imposed by the temporal component. Here, we expound on the practical implementation, demonstrating the strategic utilization of PySpark in conjunction with Python. This dynamic combination enabled seamless processes encompassing data collection, loading, thorough analysis, and, ultimately, the application of forecasting techniques to meteorological data. The subsequent sections delve into the the details of our approach, shedding light on the pivotal role played by PySpark and Python in handling and interpreting the expansive and time-sensitive datasets related to our climate analysis deliverable.

The initial phase of our analysis involves conducting an in-depth Exploratory Data Analysis (EDA) on the dataset sourced from the OpenDataSoft platform[8]. This dataset encapsulates diverse meteorological attributes pertaining to 21 French regions situated across various continents. The platform offers flexibility in selecting both the data range and the size of the dataset for download. It is important to note that the data is compiled from numerous stations within each region, utilizing an array of sensors and other cutting-edge technologies. An initial inspection of the data reveals the existence of a total of 62 distinct stations, providing a comprehensive perspective on meteorological conditions. With over 23 weather features available, some of which are extraneous to our project objectives, we conduct rigorous data analysis to discern the relevance of each feature. This process allows us to identify and retain only those features that hold significance for the subsequent phases of our work. Moreover, a notable observation is that all features are currently classified as strings and are marked to allow for the presence of null values (nullable = True). In response to this, we leverage the PySpark cast method to meticulously transform those features into the float data type. Subsequently, our analysis progresses to identify columns containing missing values and outliers. These irregularities in the data often stem from sensor malfunctions or operational glitches during the recording process. Identifying and isolating such values becomes imperative, considering their potential to disrupt the modeling phase in subsequent steps. Failure to address these anomalies can lead to instability in forecasting, underscoring the critical importance of this data refinement process.

7.1.1. Features Engineering

. Our initial undertaking involves addressing the issue of missing values, a crucial decision-making process for the integrity of our dataset. Guided by the percentage of missing values within each column, we adopt a strategic approach of either eliminating them entirely or imputing them with the mean value, depending on their prevalence. To facilitate this determination, we calculate the percentage of missing values for each column and establish a threshold that serves as the basis for either imputation (using the mean value) or deletion. A visual representation in the accompanying figure illustrates instances where certain columns exhibit a complete absence of values (100% missing). In such cases, opting for elimination becomes a logical and meaningful course of action.

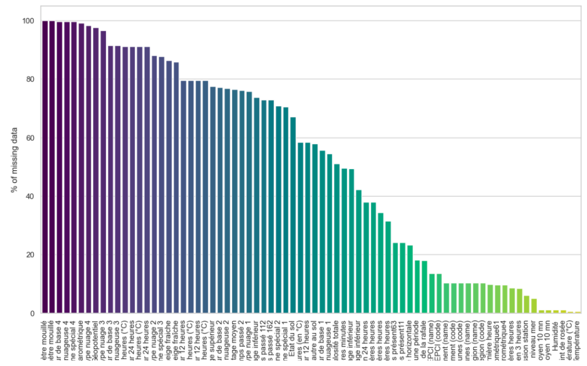
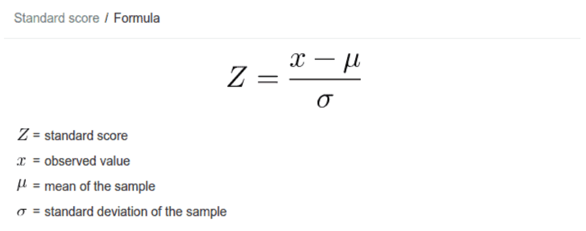


Fig. 2: Missing values

In addressing outliers within our dataset, we employ a comprehensive approach that combines visualizations and statistical measures. Initially, we use different visualizations such as distributions (for tails) and bar plots, particularly focusing on key metrics like temperature to discern the presence of outliers. Besides, we calculate the Z-score, a statistical measure that quantifies the deviation of a data point from the mean in terms of standard deviations. The Z-score is computed using the formula below. This calculation is performed for each row within the dataset, allowing us to identify and quantify outliers effectively.



Standard score / Formula

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

Z = standard score

x = observed value

μ = mean of the sample

σ = standard deviation of the sample

Fig. 3: Z-score formula

To apply the removal of outliers, we implement a function named remove_outliers_zscore. This function utilizes PySpark's capabilities to calculate the Z-score for specified

columns and subsequently filters out rows where the absolute Z-score exceeds a predefined threshold, which is set at 2 by default. The function is designed to handle multiple columns, providing flexibility in identifying and eliminating outliers across various metrics. The resulting dataset, post-application of this function, ensures a more robust and accurate representation of the meteorological data, free from the distortions introduced by outliers.
In our analysis, we start by working with a representative sample of the dataset, executing a query operation to retrieve data spanning a 10-year period. Subsequently, we employ targeted scripts to delve into data distribution and apply diverse feature engineering techniques. Having initially addressed missing data and outliers, deciding whether to remove or impute based on the specific z-score formula, our current focus shifts to understanding the interrelationships between various features. One critical aspect is exploring the correlation between different meteorological attributes. For example, it is vital to observe how changes in temperature might impact humidity. This insight aids in selecting relevant features that exhibit correlation with our target variable. When assessing correlation, we consider whether two features correlate positively or negatively, emphasizing the general coefficient as it signifies the presence and direction of the correlation. The correlation matrix in Figure 4, generated using Pearson's correlation formula, succinctly encapsulates the interconnectedness of these features, providing a visual representation that aids in our subsequent modeling and analysis.

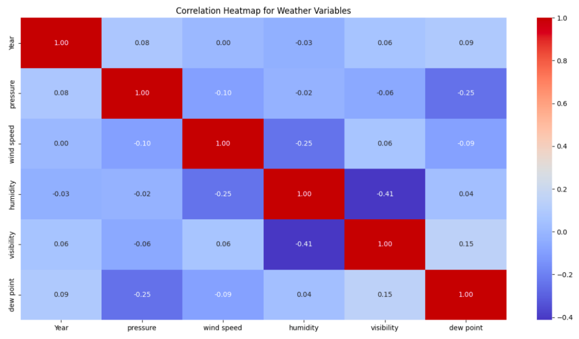


Fig. 4: Correlation Matrix

To construct the aforementioned correlation matrix, we employ a strategic blend of PySpark code to select the pertinent columns. Following this column selection, we seamlessly convert the Resilient Distributed Dataset (RDD) to a Pandas DataFrame. This conversion is imperative as it enables us to directly leverage the Pandas library's built-in correlation method. Subsequently, we make use of Seaborn, a Python data visualization library, to generate the correlation matrix. Seaborn allows us to visually represent the correlation scores between different features through a heatmap. This heatmap provides an intuitive and insightful representation of the relationships between meteorological attributes, facilitating a deeper understanding of their interdependencies.
The resulting correlation matrix serves as a valuable visual helper in our ongoing analysis and decision-making processes.

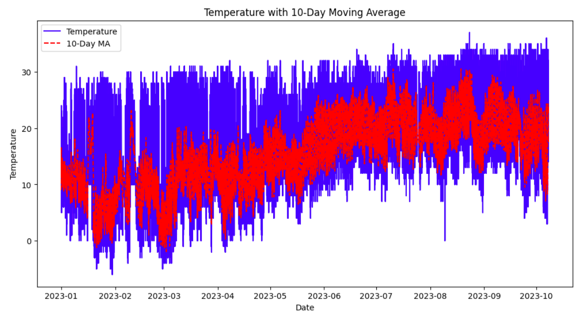


Fig. 5: Moving Average

we implement a 10-day moving average for the temperature metric. This technique holds paramount significance in smoothing out short-term fluctuations and highlighting underlying trends in the meteorological data. The 10-day moving average involves calculating the average temperature over a rolling 10-day window, providing a more stabilized representation of temperature patterns. This smoothing operation helps in revealing overarching trends and variations, mitigating the impact of daily or hourly fluctuations that might obscure meaningful insights. By incorporating this metric into our analysis, we enhance our ability to discern and interpret temperature trends over time, contributing to a more comprehensive understanding of atmospheric conditions within the targeted regions. We will add this representation to our seasonal decomposition to emphasize the need of finding trends and seasonality in time series data before applying specific forecasting models.

Furthermore, as we have a time component in the retrieved data, and since we have already considered applying a moving average to represent a smooth representation of the condensed data, understanding the different components of time series data would be crucial for accurate forecasting of the selected meteorological feature. Typically, time series data typically consists of three main components: trend, seasonality, and residuals. The trend represents the long-term progression in the data, indicating whether, for example, temperatures are generally increasing, decreasing, or remaining stable over time. Seasonality captures recurring patterns or fluctuations that follow a specific temporal cycle, such as temperature variations corresponding to different seasons throughout the year. Finally, residuals represent the random fluctuations or noise in the data that are not explained by the trend or seasonality.

First and foremost, seasonal decomposition allows us to isolate and analyze each component individually, providing a clearer understanding of the underlying patterns and variations in the meteorological data. This decomposition aids in identifying

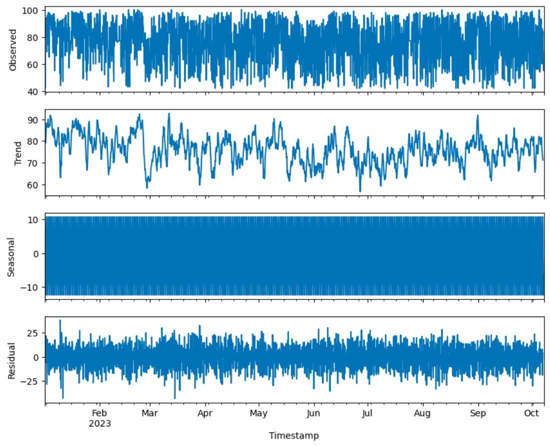
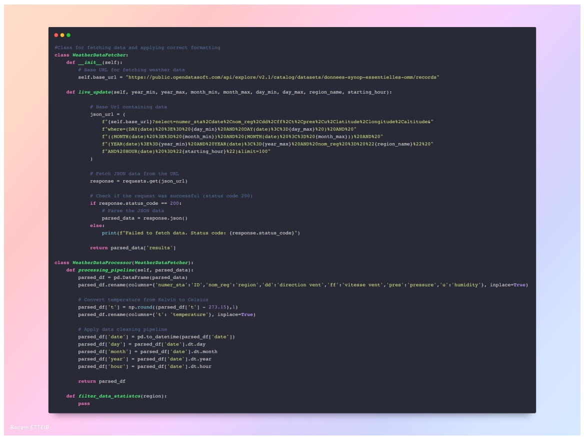


Fig. 6: Seasonal Decomposition



```python
#Class for fetching data and applying correct formatting
class WeatherDataFetcher:
    def __init__(self):
        # Base URL for fetching weather data
        self.base_url = "https://public.opendatasoft.com/api/explore/v2.1/catalog/datasets/donnees-synop-essentielles-omm/records"

    def live_update(self, year_min, year_max, month_min, month_max, day_min, day_max, region_name, starting_hour):

            # Base Url containing data
            json_url = (
                f"{self.base_url}?select=numer_sta%2Cdate%2Cnom_reg%2Cdd%2Cff%2Ct%2Cpres%2Cu%2Clatitude%2Clongitude%2Caltitude&"
                f"where=(DAY(date)%20%3E%3D%20{day_min}%20AND%20DAY(date)%3C%3D{day_max}%20)%20AND%20"
                f"((MONTH(date)%20%3E%3D%20{month_min})%20AND%20(MONTH(date)%20%3C%3D%20{month_max}))%20AND%20"
                f"(YEAR(date)%3E%3D{year_min}%20AND%20YEAR(date)%3C%3D{year_max}%20AND%20nom_reg%20%3D%20%22{region_name}%22%20"
                f"AND%20HOUR(date)%20%3D%22{starting_hour}%22)&limit=100"
            )

            # Fetch JSON data from the URL
            response = requests.get(json_url)

            # Check if the request was successful (status code 200)
            if response.status_code == 200:
                # Parse the JSON data
                parsed_data = response.json()
            else:
                print(f"Failed to fetch data. Status code: {response.status_code}")

            return parsed_data['results']

class WeatherDataProcessor(WeatherDataFetcher):
    def processing_pipeline(self, parsed_data):
        parsed_df = pd.DataFrame(parsed_data)
        parsed_df.rename(columns={'numer_sta':'ID','nom_reg':'region','dd':'direction vent','ff':'vitesse vent','pres':'pressure','u':'humidity'}, inplace=True)

        # Convert temperature from Kelvin to Celsius
        parsed_df['t'] = np.round((parsed_df['t'] - 273.15),1)
        parsed_df.rename(columns={'t': 'temperature'}, inplace=True)

        # Apply data cleaning pipeline
        parsed_df['date'] = pd.to_datetime(parsed_df['date'])
        parsed_df['day'] = parsed_df['date'].dt.day
        parsed_df['month'] = parsed_df['date'].dt.month
        parsed_df['year'] = parsed_df['date'].dt.year
        parsed_df['hour'] = parsed_df['date'].dt.hour

        return parsed_df

    def filter_data_statistcs(region):
        pass
```

Fig. 7: Data Query API

the dominant seasonal cycles and trends, helping us make informed decisions during the subsequent modeling stages. Moreover, by separating these components, we enhance the accuracy of our forecasting models. Seasonal decomposition enables us to model and forecast each component independently, allowing for more precise predictions. For instance, being able to model the seasonality component separately allows us to account for the distinct weather patterns associated with different seasons, which is vital for accurate climate predictions. By taking a closer look at the mean temperature over the years, we could observe repeated patterns every two years (same frequency) which perfectly fits in the definition of seasonality. We could use this information to make accurate forecasts based on the selected year.

7.1.2. API for real time analysis

. In order to facilitate real-time data gathering with a tailored approach, we designed a versatile API using Python. The API, encapsulated within the WeatherDataFetcher class, enables users to fetch meteorological data from the OpenDataSoft platform with specific constraints such as date range, region, and starting hour. This API leverages OpenDataSoft's open access API and utilizes a query language to dynamically tailor the data retrieval process. The code allows users to specify the desired date boundaries, region name, and starting hour, ensuring a highly customizable and interactive experience.

For instance, the live_update function within the WeatherDataFetcher class utilizes a base URL and constructs a dynamic query to fetch data in real time based on user-specified parameters. The URL is formatted to filter data based on the selected date range, region, and starting hour, ensuring a targeted and efficient data retrieval process. This functionality is crucial for our project, enabling us to collect relevant and up-to-date meteorological data that aligns with the specific needs of our analysis. Additionally, the WeatherDataProcessor class, which inherits from WeatherDataFetcher, includes a comprehensive processing pipeline to structure and clean the fetched data. This pipeline involves converting temperature from Kelvin to Celsius, renaming columns, and extracting relevant date-related information. The resulting processed DataFrame provides a refined and organized dataset for further analysis.

In essence, this API empowers our project by offering a streamlined and customizable mechanism for obtaining real-time meteorological data. It enhances the flexibility and adaptability of our data gathering process, aligning with the dynamic nature of atmospheric conditions and ensuring that our analyses are based on the most recent and relevant information. The provided example showcases how users can leverage this API to fetch and process data for a specific region, date range, and starting hour, exemplifying its practical application within the context of Climate Insights.

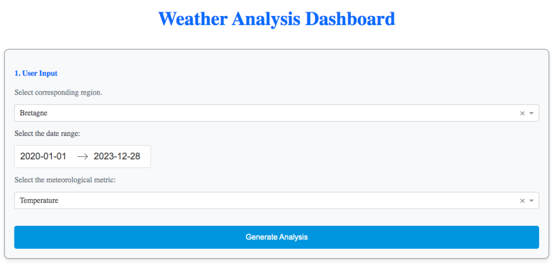
8. Production

Following the completion of Exploratory Data Analysis (EDA) on the gathered meteorological data, we transition into the modeling phase. It's noteworthy that the earlier analysis was conducted on a sample dataset using a single-core machine. In this section, we address the crucial shift towards a multi-cluster environment, specifically leveraging Google Cloud. We delve into the setup process, outlining key advantages that this transition offers compared to a single-core machine. The mitigation towards a multi-cluster environment becomes instrumental in enhancing computational efficiency, scalability, and overall performance.

Subsequently, we shift our focus to the final deliverables of the project. This encompasses an interactive user interface developed using Dash, providing real-time data analysis capabilities. The interface offers basic insights into meteorological metrics such as temperature, humidity, and pressure. Additionally, we explore the model employed for forecasting selected metrics, elucidating how it contributes to our overarching goal of understanding and predicting atmospheric phenomena. This holistic approach, spanning from multi-cluster environment implementation to the development of an interactive user interface and forecasting model, positions our project as a comprehensive and impactful solution within the domain of climate analysis.

8.1. Dash

The dashboard represents the frontend of the project. To allow the user, more specifically the weather specialist, to easily navigate and analyse the queried data, we make use of Dash, an interactive Python library, to display the different phases of the project. A possible use of the dashboard would be to select the date range for the data to be scraped, we note that the data platform is usually unavailable for updates from 7PM to 11PM, the region in question and the desired metric to be forecasted and analysed. An example of use case is illustrated by the figures below.



Weather Analysis Dashboard

1. User Input

Select corresponding region.

Bretagne

Select the date range:

2020-01-01 → 2023-12-28

Select the meteorological metric:

Temperature

Generate Analysis

Fig. 8: Dashboard section 1

The user is provided with a set of possible choices for analysis, and the data is limited to French regions. However, the project could be extended to work on global data thanks to the integration of Google Cloud for faster computations. Google Cloud offers various data analytics and integration solutions that can be leveraged for this purpose. In the next sections, we discuss the integration of Spark with Google Cloud to leverage the power of distributed processing systems.

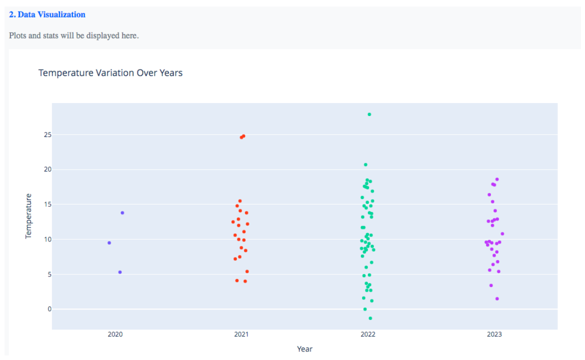


Fig. 9: Dashboard section 2

The depicted figure showcases a subset of the generated visualizations resulting from the user's input processed through the Dash input method. Specifically, the plot illustrates the temporal variation in temperature over the selected years. In this visualization, the user has chosen temperature as the metric of interest, and the x-axis corresponds to the period specified by the user. This dynamic representation provides an insightful and user-friendly means of exploring how temperature fluctuates over the specified time frame. The interactive nature of the Dash interface allows users to seamlessly tailor their analyses, gaining valuable insights into the temperature trends based on their specified parameters

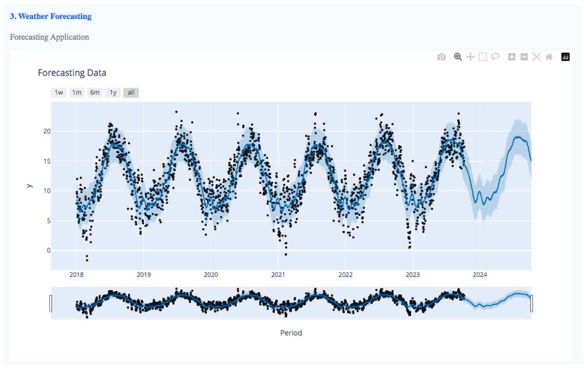
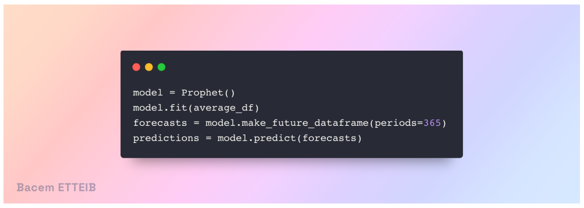


Fig. 10: Dashboard section 3

In this segment of the application, we leverage the Prophet model (see section 8.2) to perform forecasts on the selected metric values over a specified future period. The interactive capabilities of Dash further enhance the user experience by enabling zoom functionality and facilitating the selection of specific forecasted points corresponding to individual time points. This feature empowers users to closely examine and analyze forecasted values within their chosen temporal intervals.
An important aspect of our forecasting approach involves the ability to compare the model's predictions with those made by the national weather center, serving as a benchmark to assess accuracy. It's crucial to acknowledge the inherent volatility in weather forecasting due to the unpredictable nature of natural phenomena. Despite these challenges, our pre-deployment experiments have yielded promising results, particularly for close date ranges concerning temperature and humidity metrics.
It's worth highlighting that, upon user request, the forecasting capabilities extend beyond temperature and humidity to include other significant meteorological factors such as wind direction and pressure. This diversity in the climate factors and also the various regions available for the analysis implies the use of Spark to evaluate and deploy multiple time series models and applications at the same time. For instance, we could run 10 instances of the Facebook Prophet model at the same time.

8.2. Prophet Model

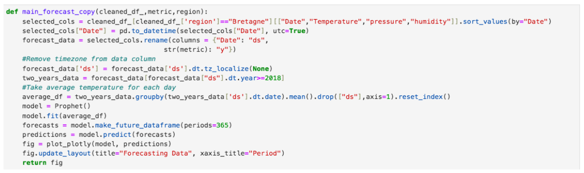
Prophet is a procedure for forecasting time series data based on an additive model. It is designed to handle data with strong seasonal effects and multiple seasons of historical data. The model is robust to missing data, shifts in the trend, and outliers, making it suitable for meteorological datasets, which often exhibit irregularities and sudden changes. Prophet is open source and available for download on CRAN and PyPI. It is implemented in both R and Python, allowing users to choose the language they are most comfortable with for forecasting. The procedure is fully automatic, providing reasonable forecasts on messy data with no manual effort. Additionally, it includes options for users to tweak and adjust forecasts, enabling the use of human-interpretable parameters to improve predictions by incorporating domain knowledge. The model is known for its speed and accuracy, outperforming other approaches in the majority of cases.[11]



```
model = Prophet()
model.fit(average_df)
forecasts = model.make_future_dataframe(periods=365)
predictions = model.predict(forecasts)
```

Fig. 11: Prophet Model

The provided figure below delineates the implementation of the forecasting function within a PySpark environment, intended to be invoked within the Dash callback function. This function receives a preprocessed dataframe, adhering to the specifications outlined in the preceding Exploratory Data Analysis (EDA) section, and a designated region for weather analysis. Given that data is collected every 4 hours within the same day, and to mitigate overfitting concerns, a deliberate decision is made to consider only the daily average of the selected metric (e.g., temperature). This strategic aggregation transforms the dataset from containing six daily metric values to a singular representative value.



```
def main_forecast_copy(cleaned_df_,metric,region):
    selected_cols = cleaned_df_[cleaned_df_['region']=="Bretagne"][["Date","Temperature","pressure","humidity"]].sort_values(by="Date")
    selected_cols["Date"] = pd.to_datetime(selected_cols["Date"], utc=True)
    forecast_data = selected_cols.rename(columns = {"Date": "ds",
                                         str(metric): "y"})
    #Remove timezone from data column
    forecast_data['ds'] = forecast_data['ds'].dt.tz_localize(None)
    two_years_data = forecast_data[forecast_data["ds"].dt.year>=2018]
    #Take average temperature for each day
    average_df = two_years_data.groupby(two_years_data['ds'].dt.date).mean().drop(["ds"],axis=1).reset_index()
    model = Prophet()
    model.fit(average_df)
    forecasts = model.make_future_dataframe(periods=365)
    predictions = model.predict(forecasts)
    fig = plot_plotly(model, predictions)
    fig.update_layout(title="Forecasting Data", xaxis_title="Period")
    return fig
```

Fig. 12: Forecasting function

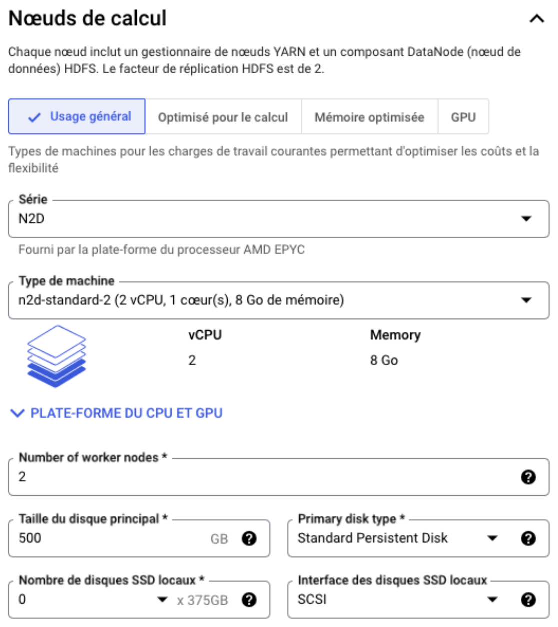
The code begins by selecting relevant columns from the cleaned dataframe for the specified region, transforming the data into a format suitable for the Prophet model. It then initializes the Prophet model, fits it to the averaged dataframe, and generates forecasts for the next 365 periods. The resulting plot, visualizing the forecasted data, is produced using Plotly.

8.3. Google Cloud setup

In this concluding section, we delve into the establishment of the Google Cloud environment. The transition from single-core to distributed processing systems becomes imperative to effectively manage the substantial volume of meteorological data intrinsic to our project. It is noteworthy that our current implementation has solely utilized a subset of the available data. As we envision expanding the scope of the project, the necessity for robust infrastructure, exemplified by Google Cloud, becomes increasingly apparent.

8.4. Dataproc Cluster

The Dataproc cluster comprises a set of interconnected virtual machines, collectively forming a computational powerhouse. The master node manages the overall coordination of tasks, while the worker nodes carry out parallelized computations, enabling the processing of extensive meteorological data. The distributed architecture of Dataproc allows for the parallel execution of tasks across multiple nodes, thereby significantly enhancing computational efficiency.



Nœuds de calcul

Chaque nœud inclut un gestionnaire de nœuds YARN et un composant DataNode (nœud de données) HDFS. Le facteur de réplication HDFS est de 2.

Types de machines pour les charges de travail courantes permettant d'optimiser les coûts et la flexibilité

Série: N2D

Fourni par la plate-forme du processeur AMD EPYC

Type de machine: n2d-standard-2 (2 vCPU, 1 cœur(s), 8 Go de mémoire)

vCPU	Memory
2	8 Go

Number of worker nodes *: 2

Taille du disque principal *: 500 GB

Primary disk type *: Standard Persistent Disk

Nombre de disques SSD locaux *: 0 x 375GB

Interface des disques SSD locaux: SCSI

Fig. 13: Dataproc cluster

Google Cloud Compute Engine provides virtual machine instances that can be customized to meet specific computational requirements. In our context, Compute Engine serves as the underlying infrastructure for the Dataproc cluster, offering scalable and configurable virtual machines for master and worker nodes, ensuring efficient distributed data processing. Introducing n2d series in Google Cloud Dataproc cluster, we opted for a configuration with two worker nodes on Compute Engine, the disk size was kept as default.

8.5. Additional Components

To enrich our virtual environment, we utilize the additional components option in Google Cloud, integrating essential addons to enhance functionality. One notable choice is the integration of Jupyter Notebook, providing an interactive platform for code manipulation. Additionally, we actively monitor our system's real-time performance, utilizing the interactive cloud interface to observe CPU utilization and disk throughput.

9. Bibliographie

References

[1] Sandy Ryza, Uri Laserson, Sean Owen, and Josh Wills. "Advanced Analytics with Spark. 2nd edition."
[2] SALLOUM, Salman, DAUTOV, Ruslan, CHEN, Xiaojun, PENG, Patrick Xiaogang and HUANG, Joshua Zhexue. Big Data Analytics on Apache Spark - International Journal of Data Science and Analytics.
[3] Almeida A, Brás S, Sargento S, Pinto FC. "Time series big data: a survey on data stream frameworks, analysis and algorithms."
[4] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, Ion Stoica. "Apache Spark: A Unified Engine For Big Data Processing".
[5] Sumit Gupta, Shilpi Saxena. Real-Time Big Data Analytics.
[6] The Climate Service Center Germany (GERICS). "Statistical methods for the analysis of simulated and observed climate data Applied in projects and institutions dealing with climate change impact and adaptation."
[7] Akhsay Hari. "Deploying PySpark Machine Learning models with Google Cloud Platform using Streamlit."
[8] Data Source. "https://public.opendatasoft.com/explore/dataset/donnees-synop-essentielles-omm/information/?sort=date".
[9] Databricks: Features Engineering using PySpark.
[10] Emanuel Parzen. "An Approach to Time Series Analysis." Institute of Mathematical Statistics 1961.
[11] Prophet model, Meta Facebook. "https://facebook.github.io/prophet/".

10. Plagiarism statement

I declare that I am aware of the following facts:

- I understand that in the following statement the term "person" represents a human or **ANY AUTOMATIC GENERATION SYTEM**.
- As a student at the University of Luxembourg I must respect the rules of intellectual honesty, in particular not to resort to plagiarism, fraud or any other method that is illegal or contrary to scientific integrity.
- My report will be checked for plagiarism and if the plagiarism check is positive, an internal procedure will be started by my tutor. I am advised to request a pre-check by my tutor to avoid any issue.
- As declared in the assessment procedure of the University of Luxembourg, plagiarism is committed whenever the source of information used in an assignment, research report, paper or otherwise published/circulated piece of work is not properly acknowledged. In other words, plagiarism is the passing off as one's own the words, ideas or work of another person, without attribution to the author. The omission of such proper acknowledgement amounts to claiming authorship for the work of another person. Plagiarism is committed regardless of the language of the original work used. Plagiarism can be deliberate or accidental. Instances of plagiarism include, but are not limited to:
  1) Not putting quotation marks around a quote from another person's work
  2) Pretending to paraphrase while in fact quoting
  3) Citing incorrectly or incompletely
  4) Failing to cite the source of a quoted or paraphrased work
  5) Copying/reproducing sections of another person's work without acknowledging the source
  6) Paraphrasing another person's work without acknowledging the source
  7) Having another person write/author a work for oneself and submitting/publishing it (with permission, with or without compensation) in one's own name ('ghost-writing')
  8) Using another person's unpublished work without attribution and permission ('stealing')
  9) Presenting a piece of work as one's own that contains a high proportion of quoted/copied or paraphrased text (images, graphs, etc.), even if adequately referenced

Auto- or self-plagiarism, that is the reproduction of (portions of a) text previously written by the author without citing that text, i.e. passing previously authored text as new, may be regarded as fraud if deemed sufficiently severe.

11. Appendix

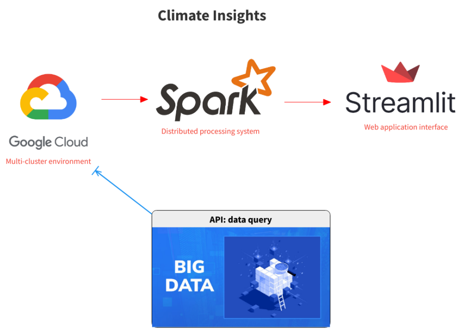

Climate Insights
Google Cloud
Multi-cluster environment
Spark
Distributed processing system
Streamlit
Web application interface
API: data query
BIG DATA

Fig. 14: Wireframe

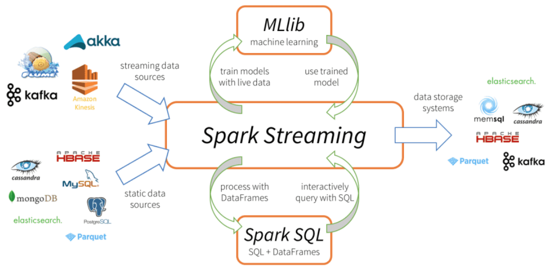


Fig. 15: Databricks Spark Streaming illustration

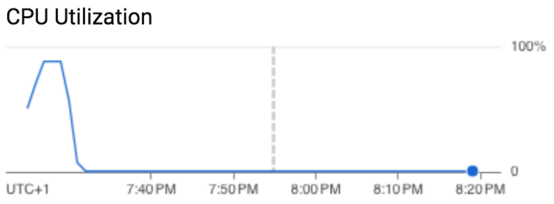


Fig. 16: CPU utilization

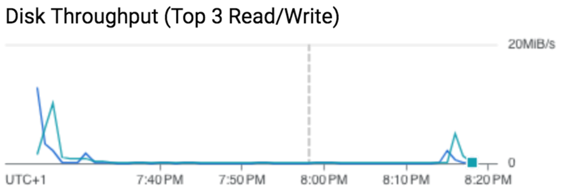


Fig. 17: Disk Throughput