



University of Piraeus

**SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGIES
DEPARTMENT OF DIGITAL SYSTEMS**

BACHELOR THESIS

**Self-Adjustable Big Data Management exploiting
the Hadoop Ecosystem**

Panagiotis Karamolegkos

Supervising Professors:

Dimosthenis Kyriazis, Associate Professor

Argyro Mavrogiorgou, PostDoctoral Academic Fellow

PIRAEUS

JULY 2022

BACHELOR THESIS

Self-Adjustable Big Data Management exploiting the Hadoop Ecosystem

Panagiotis Karamolegkos

A.M.: E17065

The road to get my Bachelor's degree was long, but there were many who stood along my side, helping me get through troublesome and stressful situations. The help I received was not money, it was something more. Good and bad memories, many smiles and lifelong friendships.

I would like to thank:

My parents, for their everyday support in my journey

My sisters, for making every day of my journey interesting and meaningful

My friends, for the memories we built together

My professors, for the opportunities they gave me

My self, for always taking one more step forward

Contents

List of Tables.....	iii
List of Figures.....	iv
Acronyms	v
Abstract.....	vi
1. Introduction	1
1.1. Purpose of the document	1
1.2. Document Structure	3
2. Bibliography Review.....	4
2.1. Big Data	4
2.2. Big Data Lifecycle	5
2.3. Big Data Collection.....	6
2.4. Big Data Storage.....	6
2.5. Big Data Processing.....	7
2.5.1. Big Data Processing Tools.....	7
2.5.1.1. MapReduce.....	9
2.5.1.2. Spark	10
2.5.2. Big Data Processing Techniques	11
2.5.3. MapReduce vs Spark	13
2.6. Big Data Analysis	15
2.6.1. Big Data Analysis Tools.....	16
2.6.1.1. Mahout.....	17
2.6.1.2. MLlib	17
2.6.2. Big Data Analysis Techniques	18
2.6.3. Mahout vs MLlib	19
3. Proposed Big Data Management Platform	21
3.1. Implementation Objectives	21
3.2. End-users	25
3.3. Tools and Frameworks.....	26
3.4. Implementation Plan	27
3.4.1. High Level Architecture Plan	27
3.4.2. Low Level Architecture Plan.....	28
3.4.3. System Interaction Plan	31
3.4.4. Execution Plan	32
4. Platform Implementation	33
4.1. Installation Guide	33

4.1.1.	Prerequisites for Installation	33
4.1.2.	Installation Process	34
4.1.3.	Execution Process	36
4.2.	Implementation Technologies.....	36
4.3.	User Manual.....	37
4.3.1.	Authentication Interface	37
4.3.2.	Homepage Interface.....	40
4.3.3.	Collection Interface	40
4.3.4.	Pre-processing Interface	41
4.3.5.	Processing Interface	43
4.3.6.	Analytics Interface.....	45
4.3.7.	Visualization Interface	48
4.3.8.	Management Interface	50
5.	Platform Evaluation	52
5.1.	Data Description	52
5.2.	Experimentation Environment	52
5.3.	Experimental Results	53
6.	Conclusions and Next Steps	58
	References.....	59

List of Tables

Table 1 – Wordcount Analysis Results	13
Table 2 – Logistic Regression Analysis Results	13
Table 3 – Hadoop MapReduce and Spark differences	14
Table 4 – Implemented Algorithms of Mahout and Spark MLlib	19
Table 5 – Implemented Algorithms of different Frameworks	19
Table 6 – EverAnalyzer Objectives.....	21
Table 7 – Correspondence of systems and Objectives.....	28
Table 8 – Diseases and conditions used	52
Table 9 – Pre-processing Results.....	53
Table 10 – Processing Results.....	54
Table 11 – Processing Execution Speeds	55

List of Figures

Figure 1 – Big Data characteristics.....	4
Figure 2 – Big Data lifecycle.....	5
Figure 3 – MapReduce Architecture.....	10
Figure 4 – Spark Architecture.....	11
Figure 5 – EverAnalyzer Logo.....	21
Figure 6 – Use case diagram of Objective #1.....	22
Figure 7 – Use case diagram of Objective #2.....	22
Figure 8 – Use case diagram of Objective #3.....	23
Figure 9 – Use case diagram of Objective #4.....	23
Figure 10 – Use case diagram of Objective #5.....	24
Figure 11 – Use case diagram of Objective #6.....	24
Figure 12 – Use case diagram of Objective #7.....	24
Figure 13 – EverAnalyzer end-users.....	25
Figure 14 – High Level Architecture.....	27
Figure 15 – Low Level Architecture.....	28
Figure 16 – Component Diagram.....	31
Figure 17 – Class diagram.....	32
Figure 18 – Sign-up interface.....	38
Figure 19 – Sign-in interface.....	38
Figure 20 – Existing User / Sign-up.....	39
Figure 21 – Wrong Credentials / Sign-in.....	39
Figure 22 – Homepage Interface.....	40
Figure 23 – Collection Interface.....	41
Figure 24 – Collection datasets.....	41
Figure 25 – Pre-processing form.....	42
Figure 26 – No collection datasets.....	42
Figure 27 – Pre-processing datasets - processing.....	43
Figure 28 – Processing form.....	44
Figure 29 – Processing proposal.....	44
Figure 30 – No pre-processed datasets – processing.....	45
Figure 31 – Pre-processing datasets - analytics.....	45
Figure 32 – Analysis form.....	46
Figure 33 – Analysis proposal.....	47
Figure 34 – No pre-processed datasets - analytics.....	47
Figure 35 – Visualization lists.....	48
Figure 36 – Visualizable results.....	48
Figure 37 – Wordcount visualization.....	49
Figure 38 – K-means visualization.....	49
Figure 39 – Management lists.....	50
Figure 40 – Management results.....	50
Figure 41 – Viewing pre-processing.....	51
Figure 42 – Viewing Wordcount.....	51
Figure 43 – Correct suggestions streak.....	56
Figure 44 – Worst-Best time difference.....	57
Figure 45 – Worst-Best EverAnalyzer.....	57

Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Networks
API	Application Programming Interface
CBM	Condition Based Maintenance
CSS	Cascading Style Sheets
DAG	Directed Acyclic Graph
DBMS	Database Management System
GPS	Global Positioning System
HDD	Hard Disk Drives
HDFS	Hadoop Distributed File System
HTML	Hypertext Markup Language
IoT	Internet of Things
IT	Information Technology
JAR	Java Archive
JS	JavaScript
JSON	JavaScript Object Notation
JSP	Java Server Pages
ML	Machine Learning
NoSQL	Non-Structured Query Language
OS	Operating System
PCA	Principle Component Analysis
RAM	Random Access Memory
RDBMS	Relational Database Management System
RDD	Resilient Distributed Dataset
SQL	Structured Query Language
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TCP	Transmission Control Protocol
UML	Unified Modelling Language
URL	Uniform Resource Locator
YARN	Yet Another Resource Negotiator

Abstract

The growth of the Internet of Things (IoT), as well as the widespread use of social media, have increased the use of internet by the population of Earth. This increase has resulted in the production of giant volumes of data that are difficult to be handled. These data, however, are containing useful information for various stakeholders. All these data is defined as Big Data. Many companies collect data from multiple sources and analyze them to improve their business decisions about their customers and their market needs and trends. To successfully analyze these data, there have been developed various Big Data analysis tools, to efficiently extract information from these large volumes of data. Among these tools, there are widely adopted and used the Apache Hadoop and the Apache Spark, which offer effective solutions for analyzing huge amounts of data. The goal of this Thesis is to design and implement an adjustable Big Data management platform, being able to automatically collect, pre-process, process, and analyze Big Data, as well as give the ability of their visualization. Because of the many different tools that have been developed to perform common processing tasks, it is important to choose the right tool for the different processing use cases. The platform can gather knowledge and suggest one of the two (2) mentioned tools that is better for the execution of the end-user's use case. In this document, the purpose of the Thesis is first mentioned. Then, a bibliographic review is done on the Big Data Lifecycle and its stages. In the literature, various tools for data processing and analysis are also analyzed. Furthermore, reference is made to techniques and other research that are relevant to this Thesis, as well as the mentioned tools. A comparison study is also done between MapReduce and Spark for data processing as well as Mahout and Spark MLlib for data analysis using machine learning algorithms. After the bibliography, follows the analysis of the proposed platform, namely EverAnalyzer, which uses the literature review to fulfill the purpose of the Thesis. This analysis provides information about the users, the architecture, the functionalities, and the way of execution, as well as a user manual of the platform. After the above, a description is given of how experimentation was performed on the platform. This experimentation was intended to examine the performance of the system that gives suggestions to users on the best tool to use based on the processing they want to perform on their data. All results were collected and compared with the ideal result that could be returned each time as a proposal. Ultimately, EverAnalyzer gave the correct suggestion 80% of the time. However, this percentage, during the experimentation, seemed to increase monotonically as the number of datasets grew. This means that while the amount of datasets within the EverAnalyzer platform is increasing, the percentage of correct answers will increase too, making it greater than 80%, as this platform learns from each analysis or processing that takes place within it using various metadata. Finally, some conclusions and next steps for further research are depicted.

1. Introduction

1.1. Purpose of the document

The growth of the Internet of Things (IoT), as well as the widespread use of social media, have increased the use of internet by the population of Earth. This increase has resulted in the production of giant volumes of data that are difficult to be handled. These data, however, are containing useful information for various stakeholders. According to Statista [1], the total amount of data being consumed globally was forecasted to increase rapidly to 64.2 Zettabytes in 2020 and 79 Zettabytes in 2021, while it is projected to grow to over 180 Zettabytes up to 2025. At the same time, Forbes [2], has predicted that more than 150 Zettabytes or 150 trillion Gigabytes of real-time data will need analysis by 2025. Multiple companies dealing with structured data need different things than the companies using unstructured data. Towards this direction, Forbes found that over 95% of companies require some help to manage the multiple sets of unstructured data.

All this data is defined as Big Data, i.e., data that can be described as large volumes of data collected from different sources and in different formats [3]. Many companies collect data from multiple sources and analyze them to improve their business decisions about their customers and their market needs and trends. To successfully analyze these data, there have been developed various Big Data analysis tools, to efficiently extract information from these large volumes of data. Among these tools, there are widely adopted and used the Apache Hadoop [4] and the Apache Spark [5], which offer effective solutions for analyzing huge amounts of data [6]. Especially Hadoop, has become one of the most renowned tools in the Information Technology (IT) industry and academic environment for its ability to manage vast amounts of data. Modern internet users are generating large unstructured datasets, increasing the demand for memory resources.

To solve the need for memory resources, a good solution is the exploitation of distributed data processing [7]. Towards this direction, two (2) of the most used tools for data processing distribution are Spark and Hadoop MapReduce, which provide to the developers who make use of them, useful functions through Application Programming Interfaces (APIs). On the one hand, MapReduce is used to process data in parallel within a processing cluster, whilst on the other hand, Apache Spark is an open-source tool built for data processing in clusters [8]. It is built on top of Hadoop and its purpose is to have a programming model that can be used in every kind of Big Data application that is limited by the functionalities of MapReduce, while being at the same time error tolerant [9]. Spark is not only an alternative to Hadoop MapReduce, but it also contains many functions for real-time data processing.

In collaboration with the tools of Mahout [10] and Spark MLlib [11], the abovementioned tools are combined to achieve Big Data analysis exploiting Machine Learning (ML) algorithms. ML is a subset of Artificial Intelligence (AI) that allows machines (i.e., systems) to learn on their own from the information they receive as an input, and make future predictions and/or suggestions [12].

Due to the continuous creation of Big Data, there are challenges that humanity is called to face, to better manage and extract useful information from the huge amounts of data produced daily. Better management of Big Data will help industries to grow and innovate, scientific research to become more optimal with more targeted results, as well as the ability to better read the present and predict the future [13]. The challenges that need to be faced refer to:

- Better management and representation of complex data formats: Big Data, refers to data whose format is not necessarily known, therefore their representation and management is difficult, as the way in which they are formatted is unknown.
- More efficient data processing: Due to their large volumes, the tools that can be used to extract information from Big Data must be designed specifically for huge datasets.
- The complexity of the Big Data management system: The systems used to process the huge volumes of data entered into them must be suitable for scaling the processes they perform efficiently.

Consequently, the purpose of this Thesis is to build a system for efficiently managing data analysis, exploiting the strengths of both the Apache Spark and the Apache MapReduce frameworks, that can be applied in the context of the Hadoop ecosystem. While there exist various implementations using Spark and MapReduce separately, the research developed in this Thesis reveals that each framework has advantages and disadvantages in different datasets. Thus, the proposed system is able to decide, based on the entered data, which of the two (2) frameworks should be preferably used, to efficiently process the underlying data. Depending on the chosen framework, the system then exploits either the Mahout (for the case of MapReduce) or the MLlib (for the case of Spark) tools to finally analyze the collected data. To successfully accomplish such processing and analysis, the system also includes a plethora of tools that can be used by the end-user for storing, managing, pre-processing the collected data, also visualizing all the captured results. For completing these tasks, the system makes use of Apache's HDFS, YARN, Kafka and Flume as well as MongoDB [4][14][15][16][17]. The collection of the used data is done through the Twitter API [18], which gives access to the Twitter database and allows the system to collect in real-time various Tweets for their further analysis.

Hence, the final goal of this Thesis is to design and implement an adjustable Big Data management platform, being able to automatically collect, pre-process, process, and analyze real-time (i.e., streaming) data. As a result, a product is developed, namely EverAnalyzer, which has the ability to better assist Data Analysts both in processing and in analysis tasks that they want to perform, reducing the workload of the end-user. To achieve all this, the system is evaluated through an experimentation that tests EverAnalyzer's ability to give empirical suggestions to its users, regarding the framework they should choose for the processes they want to perform. In the experimentation, a collection was carried out through the 30 different datasets related to different diseases and conditions in the health sector. Afterwards, pre-processing, processing and analyzes were carried out on the data. Before each analysis and processing, EverAnalyzer was asked which framework would give the shortest execution time for the process the user wanted to perform. All results were collected and compared with the ideal result that could be returned each time as a proposal. Ultimately, EverAnalyzer gave the correct suggestion 80% of the time. However, this percentage, during the experimentation, seemed to increase monotonically as the

number of datasets grew. This means that a larger number of datasets within the EverAnalyzer platform will cause an increase in the percentage of correct answers, making it greater than 80%, as this platform learns from each analysis or processing that takes place within it using various metadata.

1.2. Document Structure

This document is divided in six (6) discrete Sections:

- **Section 1 – Introduction:**
 - This Section makes an Introduction to the overall goal of the Thesis. It describes the state of Big Data in the modern world, outlining their core characteristics, as well as the challenges that need to be faced for their better and more efficient management. Also, widely used tools are referred for the successful analysis of Big Data. In the end, the Section outlines the purpose of this Thesis.
- **Section 2 – Bibliography Review:**
 - This Section provides a detailed overview of the literature review that performed for analyzing useful concepts for the Thesis, referring to Big Data. Firstly, the concept of Big Data is described and then there is an analysis on the Big Data lifecycle. Next, there is a further analysis based on the parts of Big Data lifecycle that are used in this Thesis. Finally, a comparison is carried out for Big Data Processing and Analysis Tools as well as related techniques done worldwide using such tools.
- **Section 3 – Proposed Big Data Management Platform:**
 - This Section illustrates an analysis of how the proposed platform (EverAnalyzer) is designed/built. It explains the system's implementation plan, containing its architecture, the System Interaction Plan, which refers to how the system interacts with all its components, and the Execution Plan, to clarify the procedures performed by the end-users when using the system. Also, A thorough reference is made to the system's goals and end-users, as well as the tools that is has exploited for supporting the various underlying functionalities.
- **Section 4 – Platform Implementation:**
 - This Section initially gives a User Manual in order to present how the EverAnalyzer works from the end-user's point of view. Also, an installation guide is provided, giving all the information needed to install and run the system. This guide contains information like the version of each tool used, where to find the needed repositories online and step by step guides for the installation.
- **Section 5 – Platform Evaluation:**
 - This Section analyzes the experimentation results that performed on EverAnalyzer, giving an interpretation in the end of the results that were exported.
- **Section 6 – Conclusions and Next Steps:**
 - This Section contains the conclusions of the Thesis and describes the next steps regarding the systems that can be built due to EverAnalyzer's development. Also, it describes future experiments that would be interesting to be performed following the design and implementation guidelines of EverAnalyzer.

2. Bibliography Review

2.1. Big Data

As described in Section 1, data that can be described as large volumes of data collected from diverse sources and in different formats are called Big Data [3]. Big Data can be structured, semi-structured or unstructured, making it difficult to be managed by traditional Relational Database Management Systems (RDBMS), as such solutions involve high cost and time required [19]. Structured data is the data that the information they contain and the way they contain it are known. In the other hand, semi-structured are those that are missing some specifications about the information contained inside of them and unstructured are those that their structure is not known at all. Large quantities of these data can be collected through sources such as mobile phones, social media, sensors, Global Positioning System (GPS) signals, etc. Specifically social media such as Facebook, Twitter, etc. produce a lot of data every second from the large quantities of users who use them in their daily lives [19]. Hence, Big Data may refer either to batch data deriving from ready to use datasets that need to go through some processing or analytic activities, or to streaming data coming from live sources that are always streaming information. Streaming data can be gathered and be used as batch data or they can be analyzed on the spot while they are coming from the source based on the previous data collected.

Regardless of the source producing Big Data, the latter always is accompanied by some specific characteristics, which are categorized into names with the initial letter V:

- Volume – The size of the data.
- Variety – The different formats.
- Velocity – The speed of data production.
- Veracity – The size of the data authenticity.
- Validity – The validity of the data.
- Volatility – The time frame over which the data is valid.
- Value – The usefulness of the data in terms of analysis.



Figure 1 – Big Data characteristics

The above characteristics show that Big Data are hard to be handled but can be of extremely valuable. By collecting Big Data, companies are able to analyze them and extract useful information based on their products and their clients. However, most of the time their analysis is a difficult and time-consuming process due to their shape and size diversity. At the same time, people use the internet on a daily basis more and more to facilitate their daily life and improve their entertainment, making the amount of data that are generated to increase every year.

2.2. Big Data Lifecycle

The use of Big Data can be described in a lifecycle that captures all their phases from their collection to their destruction [20]. The mentioned phases are as follows:

- **Collection:** The Collection phase is the process that the data are getting collected from different sources (data collection), most of the times in formats not easy to be handled because of their unstructured nature.
- **Storage:** In this phase, data are getting pre-processed and then are stored for their next phases. Pre-processing is of great importance (data pre-processing). This is because of the unknown structure of the collected datasets. Pre-processing gives them a standard structure that makes them easier to be handled by the next phases.
- **Analytics:** Analysis consists of algorithms useful for the extraction of results from the stored data. These algorithms consist of different processes (data processing) and analysis (data analysis). For the analytics of Big Data, visualization tools are used, giving the ability of prettier representations of the end results (data visualization).
- **Utilization:** After the Analytics phase, the Utilization of the results takes place. This is the phase where the results of Big Data are implemented in different real life and testing scenarios.
- **Destruction:** The last phase is the most important of all. In the Collection phase many sensitive data are collected from different sources. This means that there must be a strict protocol that is followed so that the confidentiality, integrity, and availability of the data is not compromised.

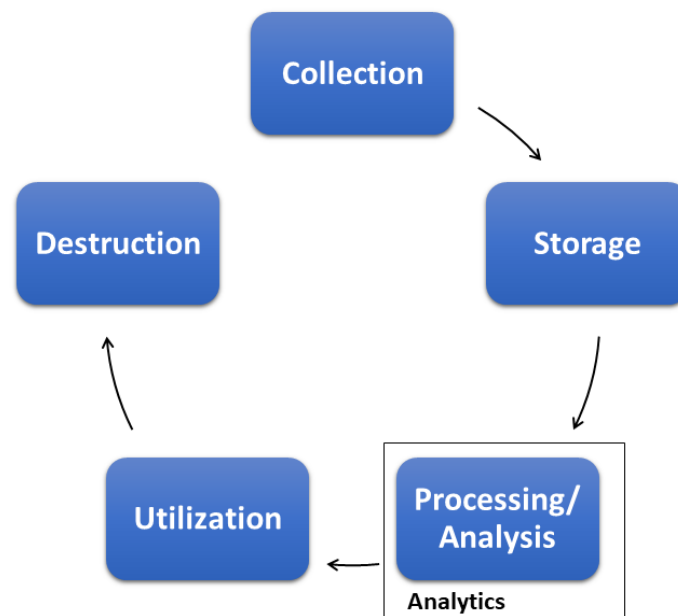


Figure 2 – Big Data lifecycle

2.3. Big Data Collection

In the review [21] data collection is described as the process of collecting data to further analyze them and get valuable results. These data can be gathered with traditional methods like questionnaires and interviews, but currently there exist many greater ways to collect data. Web services provided by different data vendors, as well as devices with sensors such as mobile phones and smart transportation cards provide data for collection.

There are two (2) basic ways to collect Big Data [22]:

- **Collection as Batch Data:** In this way the data is collected up to a predefined size and finally stored all together to be later analyzed as a set of data (Batch Processing).
- **Collection as Streaming Data:** In this way the data is analyzed during their collection. The difference between this method of collection and Batch Data is that this method applies directly Stream Processing instead of Batch Processing. Batch Processing requires a predefined amount of data to be collected, while Stream Processing will analyze each imported data directly.

Also, in case there is no way to collect enough data for some analytical process, then there are ways to construct synthetic data [23]. This data represents the real data that an analysis would probably have, but it is difficult to build good algorithms to generate them. This difficulty occurs because it is difficult to approach the real data of a population by constructing it through some system.

Based on the HoneyNet Project [24], some tools for data collection are Sebek, Hflow, Honeywall, Nepenthes, Kojoney, and Capture-HPK. Additional used for data collection purposes are Kafka and Flume. Kafka is a tool used for Streaming Data. Flume on the other hand is used to manage infrastructures on how to make channels for the collection of Streaming Data as Batch Data using Agents. These tools are combined in [25] making the tool Flafka. Flafka is giving the opportunity to save Streaming Data as Batch Data using Kafka and Flume, but this is not the reason that it was made. The reason behind Flafka existence is the need for better ways to handle event processing while having a tool whose programming is easy, making it very handy.

2.4. Big Data Storage

As described in [26], Big Data Storage refers to the storage and management of large-scale datasets, while achieving reliability and availability of data accessing. Big Data Storage mainly concerns the part of the infrastructure of the system that wishes to implement it. On the one hand, the storage infrastructure needs to provide storage services with reliable storage space. On the other hand, it must provide a powerful access interface for query and analysis of a large amount of data.

Due to the rapidly increasing amount of Big Data, powerful systems are increasingly used to store and manage them (DBMS). RDBSs are differentiated by Structured Query Language (SQL) systems and Non SQL (NoSQL) systems [27]. Based on the second survey NoSQL systems are better in use cases for storing and managing Big Data. This is because SQL systems need data to be structured to be efficient unlike NoSQL systems that are made for unstructured data.

The survey continues by separating NoSQL DBMSs in the following classes and some of the examples below:

- **Key-value Stores:** Redis, Scalaris, Tokyo Tyrant, Riak
- **Document Stores:** SimpleDB, CouchDB, MongoDB, Terrastore
- **Extensible Record Stores:** Bigtable, HBase, HyperTable, Cassandra

There is also a larger analysis of the DBMS named MongoDB. MongoDB is a schema-free document database written in C++ and developed in an open-source project. According to its developers the main goal of MongoDB is to close the gap between the fast and highly scalable key-value-stores and feature-rich traditional Relational Database Management Systems (RDBMS).

MongoDB databases reside on a MongoDB server that can host more than one of such databases which are independent and stored separately by the MongoDB server. A database contains one or more collections consisting of documents. In order to control access to the database a set of security credentials may be defined for databases.

2.5. Big Data Processing

Big Data processing consists of a set of techniques to access large-scale data to extract useful information for supporting and providing decisions, as said in [28]. Big Data processing is using various algorithms, able to be distributed among vast clusters of processing units, like Wordcount and String match [29].

Most of the times, the algorithmic complexity of the data processing algorithms is small, making them fast to execute. Their fast nature makes them easy to be implemented in order to understand different datasets. Because of their speed, they can be applied on any dataset regardless of the dataset's size.

Datasets that come directly from their source usually cannot be passed directly through a data processing job (raw datasets). This happens because datasets of this type do not have a specific structure due to the fact that they come from many different sources in the case of Big Data. In order to be able to go through a data processing job, data must therefore go through a data pre-processing phase in which their structure will be normalized. After the normalization of the data structure, it will be easy to process them by the given data processing algorithms.

2.5.1. Big Data Processing Tools

Big Data are usually stored in thousands of commodity servers so traditional programming models cannot handle them effectively. Therefore, new parallel programming models are utilized to improve the performance of Non-Structured Query Language (NoSQL) databases in datacenters. Some tools used for this job are the following [28]:

- **MapReduce:** One of the most popular programming models for Big Data processing using large-scale commodity clusters, being developed as one of the core components of the Hadoop ecosystem. MapReduce was proposed by Google and developed by Yahoo. Map and Reduce functions are programmed by users to process the Big Data distributed across multiple heterogeneous

nodes. The main advantage of this programming model is simplicity, so users can easily utilize that for Big Data processing [30].

- **Apache Pig:** Apache Pig is a Structured Query Language (SQL)-like environment developed at Yahoo, being used by a variety of organizations like Yahoo, Twitter, AOL, and LinkedIn [31].
- **Hive:** Hive is another MapReduce wrapper developed by Facebook. It provides a better environment and makes the code development simpler, since the programmers do not have to deal with the complexities of MapReduce coding [32].

Also, there are many tools build to overcome Hadoop's MapReduce issues such as slowness to load data and the lack of reuse of data. Some of these tools are:

- **Starfish:** A Hadoop-based framework, which aimed to improve the performance of MapReduce jobs using data lifecycle in analytics. It also uses job profiling and workflow optimization to reduce the impact of unbalance data during the job execution. Starfish is a self-tuning system based on user requirements and system workloads without any need from users to configure or change the settings or parameters [33].
- **Spark:** Developed at the University of California at Berkeley, Spark is an alternative to Hadoop MapReduce, which is designed to overcome the disk I/O limitations and improve the performance of earlier systems. The major feature of Spark that makes it unique is its ability to perform in-memory computations. It allows the data to be cached in memory, thus eliminating the Hadoop's disk overhead limitation for iterative tasks [34].

Similarly to MapReduce, there exist other programming models that can be used for Big Data processing in datacenters such as:

- **Dryad:** A distributed execution engine to run Big Data applications in the form of Directed Acyclic Graph (DAG). Operation in the vertexes will be run in clusters where data will be transferred using data channels including documents, Transmission Control Protocol (TCP) connections, and shared memory. Moreover, any type of data can be directly transferred between nodes. While MapReduce only support single input and output set, users can use any number of input and output data in Dryad [35].
- **Pregel:** Pregel is used by Google to process large-scale graphs for various purposes such as analysis of network graphs and social networking services [36].

Finally, there are also existing tools for data processing on Streaming Data. These tools are making processing workflows while the data are getting collected by from their source without the need to convert data to Batch Data. Such tools are the following [37]:

- Apache Storm [38]
- Apache Flink [39]
- Apache Spark (Spark Streaming) [40]
- Apache Samza [41]
- Apache Apex [42]
- Google Cloud Dataflow [43]

2.5.1.1. MapReduce

As described in [44], MapReduce is a popular Big Data processing model that has rapidly been studied and applied by both industry and academia. The purpose of MapReduce is to give the ability to process Big Data distributed across multiple heterogeneous nodes.

MapReduce has two (2) major advantages:

- It hides details related to the data storage, distribution, replication, load balancing etc. of the datasets.
- It only needs two (2) functions to be specified, which are the map function and the reduce function, making it easy to be used.

While MapReduce is referred to as a new approach of processing Big Data in cloud computing environments, it is also criticized as a “major step backwards” compared with Database Management Systems (DBMS). As the debate continues, the result shows that neither of them is good at what the other does well, and the two (2) technologies are complementary. There are systems like Aster, HadoopDB and Greenplum that mostly are still database, which simply provide a MapReduce front-end to a DBMS.

In the review [30], the architecture of MapReduce is analyzed. The MapReduce has three (3) major parts, including Master, Map function and Reduce function. The Master is responsible for managing the back-end Map and Reduce functions and offering data and procedures to them. A MapReduce application contains a workflow of jobs where each job applies one Map and one Reduce user-specified functions. The Map function is applied to each input record and produces a list of intermediate records. The Reduce function is applied to each group of intermediate records with the same key and produces a list of output records. A MapReduce program is expected to be done on several computers and nodes. Therefore, a master node runs all the necessary services to organize the communication between Mappers and Reducers. The input files are separated into the same parts called input splits. They pass to the Mappers in which they work parallel together to provide the data contained within each split. As the data is provided by the Mappers, they separate the output. Then each Reducer gathers the data partition by each Mapper, merges them, processes them, and produces the output file.

The main phases of MapReduce are:

- **Mapper:** In this phase the Mapper processes all the input data that are assigned by the master to perform some computation on this input and produce intermediate results in the form of key/value pairs.
- **Reducer:** The Reduce function receives an intermediate key and a set of values of the key. It combines these values together to form a lesser set of values.
- **Shuffle:** In MapReduce framework, after the Map task is finished, there are usually large amounts of middle data to be moved from all Map nodes to all Reduce nodes in the shuffle phase, the shuffle transfer data from the Mapper disks rather than their main memories and the intermediate result will be sorted by the keys so that all pairs with the same key will be grouped together and it needs to transfer the data from the local Map nodes to Reduce nodes through the network.

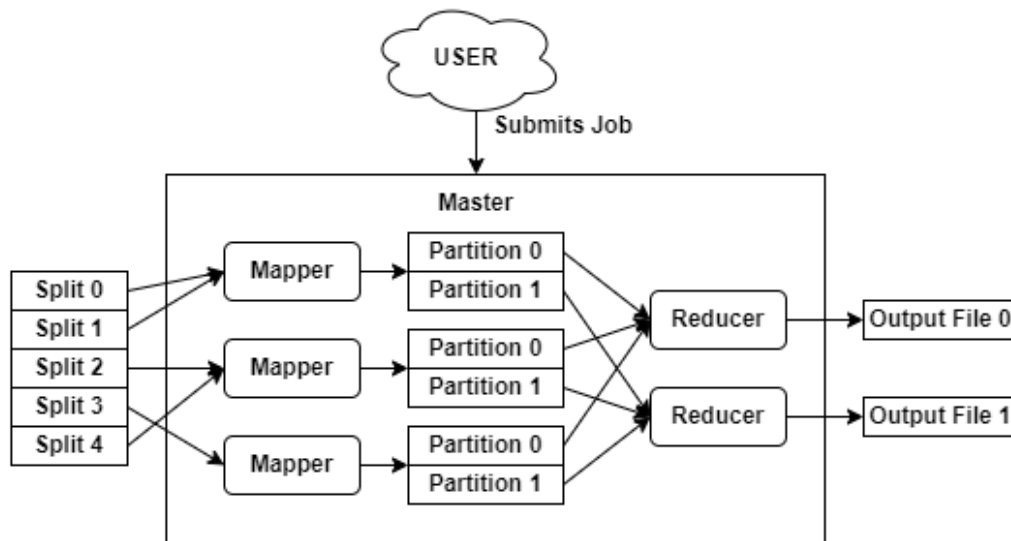


Figure 3 – MapReduce Architecture

2.5.1.2. Spark

As outlined in [34], Spark is proposed as a framework that supports large-scale data-intensive applications on commodity clusters, while retaining the scalability and fault tolerance of MapReduce. To achieve these goals, Spark introduces an abstraction called Resilient Distributed Datasets (RDD). An RDD is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost.

To use Spark, developers write a driver program that implements the high-level control flow of their application and launches various operations in parallel. Spark provides two (2) main abstractions for parallel programming: RDDs and parallel operations on these datasets.

Several parallel operations can be performed on RDDs:

- **Reduce:** Combines dataset elements using an associative function to produce a result at the driver program.
- **Collect:** Sends all elements of the dataset to the driver program. For example, an easy way to update an array in parallel is to parallelize, map and collect the array.
- **Foreach:** Passes each element through a user provided function. This is only done for the side effects of the function (which might be to copy data to another system or to update a shared variable as explained below).

The elements of an RDD do not exist in physical storage. instead, a handle to an RDD contains enough information to compute the RDD starting from data in reliable storage. This means that RDDs can always be reconstructed if nodes fail. In Spark, each RDD is represented by a Scala object. Spark lets programmers construct RDDs in four ways:

- From a file in a shared file system, such as the Hadoop Distributed File System (HDFS).
- By “parallelizing” a Scala collection (e.g., an array) in the driver program, which means dividing it into a number of slices that will be sent to multiple nodes.

- By transforming an existing RDD. A dataset with elements of any type can be transformed into a dataset with elements of another type using an operation called flatMap, which passes each element through a user-provided function of type the first type to a list of the second type.
- By changing the persistence of an existing RDD. By default, RDDs are lazy and ephemeral. That is, partitions of a dataset are materialized on demand when they are used in a parallel operation (e.g., by passing a block of a file through a map function) and are discarded from memory after use.

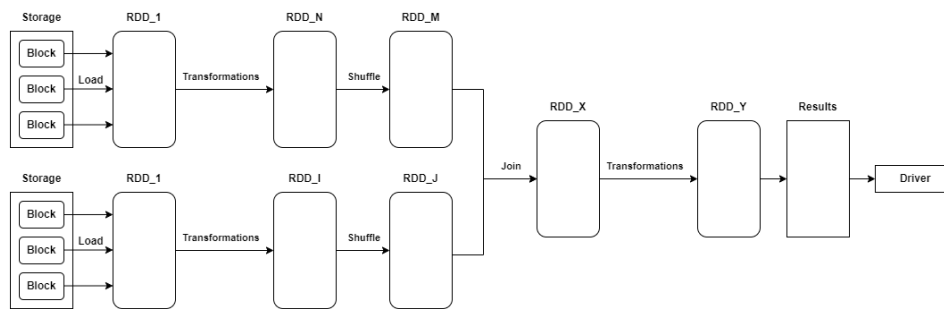


Figure 4 – Spark Architecture

2.5.2. Big Data Processing Techniques

In 2018, Aziz et al. [45] used the Spark platform to analyze Twitter data. Spark platform was analyzed for the Tweets in a short period of time, which is less than one second. Through this study, the author has focused on the executions of Hadoop MapReduce and Apache Spark, while conducted experimental simulations on the two (2) frameworks. Moreover, a comparability of two (2) executions regarding performance and architecture with an analysis to characterize the simulations' conclusions is introduced and the deficiency and drawbacks of applying Hadoop for real-time preparation are discussed as well. It is shown that a useful instrument for Streaming Data in real-time is Spark.

Pirzadeh, in his thesis [46], studied time consumed in different parts of the Hadoop platform. In order to understand the behavior of the platform, considering the bottlenecks for effective implementation, some of the criteria were simulated and analyzed. Mavridis and Karatza [47] did research on Hadoop and Spark with log file analysis, and novelty of their research was the performance evaluation of log file analysis with Hadoop and Spark. In this paper, they have investigated log file analysis with the cloud computing frameworks Apache Hadoop and Apache Spark. In both frameworks, the authors have improved the analysis of applications to realistic log file and in real Apache Web Server log files, SQL-type queries are carried out. Furthermore, with various parameters, they have led different experiments to compare and study the act and performance of the two structures and frameworks.

A project on MapReduce/Hadoop basis platform for protecting Big Data health analysis is suggested by Kuo et al. [48]. Through this research, the useful experience in implementing and designing a platform with MapReduce/Hadoop structure for protecting Big Data health analysis is reported.

In 2017, Hazarika et al. [49] examined both Spark and Hadoop platforms to study theoretical differences and their functional comparison. Their work results indicate that Spark is much faster for its cache due to its duplicate queries such as logistic

regression. Also, the limited cache weakens Spark performance for nonrepetitive queries. However, it is much faster for small repetitions from the Hadoop.

In the paper [50] several big data processing technics are introduced from system and application aspects. Cloud data management and big data processing mechanisms, are considered, including a cloud computing platform, a cloud architecture, a cloud database and a data storage scheme. Following the MapReduce parallel processing framework, it is introducing MapReduce optimization strategies and applications.

In another paper [51], it is attempted to address the issue of detecting anomalies in real time Big Data processing. This paper has surveyed the state-of-the-art real-time Big Data processing technologies related to anomaly detection. First, it is explaining the essential contexts and taxonomy of real-time Big Data processing and anomalous detection, followed by the review of Big Data processing technologies. Finally, it makes a discussion about the challenges of real-time Big Data processing in anomaly detection.

In the research [52], it is given an overview of computing infrastructure for Big Data processing, focusing on architectural, storage and networking challenges of supporting Big Data. It is discussing emerging computing infrastructure and technologies that are promising for improving data parallelism, task parallelism and encouraging vertical and horizontal computation parallelism.

Research [53] is reviewing and discussing the mechanisms of handling and processing Big Data in the Healthcare domain as well as providing a deep analysis of their applied mechanisms. It is presenting a systematic and comprehensive study about cloud computing application in the Healthcare Big Data sector. In addition, this research study offered implications to research and practice as the direction of future study for decision makers of the healthcare sector. Based on the obtained results of this study, there is adequate evidence to recommend that cloud computing can bring important benefits and opportunities to the healthcare sector.

Paper [54] investigates a practical case of a Hadoop-based medical Big Data processing system to intelligently process medical Big Data and uncover some features of hospital information system user behaviors. This paper studies user behaviors regarding various data produced by different hospital information systems for daily work. Also, in this paper it is built a five-node Hadoop cluster to execute distributed MapReduce algorithms. The distributed algorithms show promise in facilitating efficient data processing with medical Big Data in healthcare services and clinical research compared with single nodes.

Based on the performed research, this Thesis exploits both the MapReduce and the Spark frameworks, since it is based on projects and research using Streaming Data and Batch Data. The analysis of this data will be performed with a combination of Hadoop's MapReduce and Spark frameworks, constructing a system capable of selecting, through data-driven decisions, the appropriate processing framework for the input process.

2.5.3. MapReduce vs Spark

Because of the many different tools that have been developed to perform common processing tasks, it is important to choose the right tool for the different processing use cases. When it comes to data processing in processing clusters, the two (2) most commonly used tools are Apache Hadoop MapReduce [55] and Apache Spark [5].

The MapReduce tool runs within the Hadoop environment (as its name implies), usually using the Yet Another Resource Negotiator (YARN) environment [14] as cluster administrator and HDFS [4] as a distributed file storage system [56].

Spark converts input data into RDDs that allow developers to make calculations using the RAM of large computing clusters with error-tolerant ways [57]. Therefore, to minimize the cost of time, Spark mainly uses the memory of the system on which it is installed, to avoid as much as possible its communication with the Hard Disk Drives (HDD) [56].

The research [49] explains the different architectures of the aforementioned frameworks, as well as their differences in performance in two (2) experimental scenarios.

In the first scenario, programs for Wordcount analysis were used in order to compare the difference in speed between MapReduce and Spark. The results on the execution speed of the two (2) tools are shown in Table 1.

Table 1 – Wordcount Analysis Results

Number of Words	MapReduce (in sec)	Spark (in sec)
100	79	28.841
1000	91	31.185
10000	96	35.181
100000	103	36.969
1000000	116	39.569

In the second scenario, programs for Logistic Regression analysis were used comparing again the difference in the speed of the aforementioned frameworks. The results on the execution speed of the two frameworks are shown in Table 2.

Table 2 – Logistic Regression Analysis Results

Number of Iterations	MapReduce (in sec)	Spark (in sec)
5	2.541	0.9030
10	3.370	1.459
50	6.420	2.840
100	9.383	3.452
200	10.100	5.749

The research, concluded in Table 3, regarding the differences in the operation of Hadoop MapReduce and Spark.

Table 3 – Hadoop MapReduce and Spark differences

MapReduce	Spark
For Applications that repeatedly reuse the same set of data, map reduce is very inefficient.	Spark uses in-memory processing, reusing it for faster computation.
MapReduce is quite faster in batch processing.	As memory size is limited, it would be quite slower in batch processing of huge datasets.
Data is stored in disk for processing.	Data is stored in main memory. As it is an in-memory computation engine entire data is copied.
Difficulty in processing and modifying data in real-time due to its high latency.	Used to process and modify data in real-time due to its low latency.
Predominantly used to process from bygone datasets.	Predominantly used for streaming, batch processing and ML.
For fault tolerance, Map Reduce uses replication.	For fault tolerance, Spark uses RDDs.
It merges and partitions shuffle files.	It does not merge and partition shuffle files.
Primarily disk-based computation.	Primarily RAM based computation.

Relevant research was also done between the Wordcount and Tera-Sort processes in nine-node clusters with datasets reaching up to 600 Gigabytes [6]. The research showed that the performance of the Hadoop and Spark systems was largely based on the size of the data entered and the configuration given to them. Spark performed twice as well as Hadoop in the analysis of Wordcount processes and 14 times with Tera-Sort respectively. The research also agreed with the initially reported research that Spark is faster and more stable due to the processing it does based on the memory of the system on which it runs, but still agreed that for very large datasets, Spark lost gradually its speed relative to Hadoop.

Extensive research was performed using different HiBench procedures [9]. Analytical procedures were performed within a processing cloud to compare Hadoop MapReduce and Spark environments. In almost every experiment performed, Spark appeared to overshadow Hadoop with statistics reaching 92.25% increased efficiency, consuming up to 10% more memory in some cases. Spark did not seem to perform as well as Hadoop only in cases in which system memory was limited making it contact with the HDDs more often. Therefore, these results correlate with those of the above studies that have reported that Spark stops perform better than Hadoop in large datasets, having limited RAM.

2.6. Big Data Analysis

As described in [58] data analysis as a multidisciplinary concept, is defined as “the means to acquire data from diverse sources, process them to elicit meaningful patterns and insights, and distribute the results to proper stakeholders”.

Data analysis is divided into four (4) types of analysis which are [58]:

- **Descriptive analytics:** Answers the question “What has happened?” and extracts information from raw data.
- **Diagnostic analytics:** An extension of Descriptive analytics. Reports the past but tries to answer the questions like “Why did it happen?”.
- **Predictive analytics:** Answers the questions “What will happen?” and “Why will it happen?” in the future.
- **Prescriptive analytics:** It prescribes one or more courses of actions and shows the likely outcome/influence of each action. It answers the questions “What should I do?” and “Why should I do it?”. It is purely built on the “what-if” scenarios.

For the types of analysis mentioned above, there are techniques that can be used to answer the questions that each type is asked to provide. These techniques are the following [59]:

- **Clustering:** Useful for exploring data and finding natural groupings. Members of a cluster are more like each other than they are like members of a different cluster. Common examples include finding new customer segments and life sciences discovery.
- **Classification:** Most used technique for predicting a specific outcome. An example of an outcome could be knowing whether a consumer will likely buy or not a product.
- **Association:** Finds rules associated with frequently cooccurring items, used for market basket analysis, cross-sell, root cause analysis. Useful for product bundling, instore placement, and defect analysis.
- **Regression:** Technique for predicting a continuous numerical outcome such a customer lifetime value, house value, process yield rates.
- **Attribute Importance:** Ranks attributes according to strength of relationship with target attribute. Use cases include finding factors most associated with customers who respond to an offer, factors most associated with healthy patients
- **Anomaly Detection:** Identifies unusual or suspicious cases based on deviation from the norm. Common examples include health care fraud, expense report fraud, and tax compliance.
- **Feature Extraction:** Produces new attributes as linear combination of existing attributes. Applicable for text data, latent semantic analysis, data compression, data decomposition and projection, and pattern recognition.

2.6.1. Big Data Analysis Tools

There exist various data analysis tools, including open-source tools such as [60]:

- **Orange:** A tool for data analysis and visualization. Data mining is done through python or visual programming which has components for ML feature selection, and text mining [61].
- **R:** A free open-source software programming language and software environment for statistical computing and graphics. It is widely used among data miners for developing statistical software and data analysis [62].
- **Weka:** An open-source data mining software, including a collection of ML algorithms for data mining tasks such as data pre-processing, data classification, data regression data clustering, data association rules, and data visualization. The algorithms can either be applied directly to a dataset or called from JAVA code [63].
- **Shogun:** A free open-source software toolbox written in C++. It offers lots of algorithms, and data structure for ML problems. It focuses on Support Vector Machine (SVM), regression, and classification data mining problems [64].
- **Rapid Miner:** A tool operating through visual programming and is capable of manipulating, analyzing, and modelling data [65].

In addition to open-source tools, there are commercial data mining tools such as [60]:

- **Neural Designer:** A desktop application for data mining that uses neural network and ML [66].
- **SharePoint:** SharePoint is a Microsoft-hosted cloud service that empowers companies to store, access, share, and manage documented information from all devices [67].
- **Cognos:** IBM Cognos is a set of smart self service capabilities that enable them to determine and make decisions on insight quickly and confidently [68].
- **BOARD:** Board is a Management Intelligence Toolkit that combines compact software. BOARD enables users to collect and gather data from almost any source, as well as create full self-service reporting. These reports can be delivered in different formats if needed, like CSV, HTML and more [69].

While the aforementioned tools are built to perform and visualize analytics, they are not built on Big Data logic. Tools with such a way of construction are the following [60]:

- **Sisense:** Sisense is a business intelligence platform with the ability to join, analyze, and picture out information [70].
- **KEEL:** KEEL aims to help users assess evolutionary algorithms for data mining problems like regression, classification, clustering, and pattern mining. It includes a large collection of existing algorithms that it uses to compare and with new algorithms [71].
- **Mahout:** An Apache project offering algorithms for clustering, classification and batch-based collaborative filtering that run on top of Apache Hadoop [10].
- **MLlib:** Another Apache project similar to Mahout but with a different approach. MLlib runs on top of the Spark framework and makes extensive use of system's Random Access Memory (RAM) to complete the executed algorithms [72].

However, it should be noted that the two (2) most commonly used of the above tools in terms of Big Data are Mahout and Spark MLlib.

2.6.1.1. Mahout

As stated in [73], datacenter switching fabrics have enormous bandwidth demands due to the recent uptick in bandwidth-intensive applications used by enterprises to manage their exploding data. Mahout augments this basic design. It has low overhead, as it monitors and detects big flows (elephant flows) at the end-host via a layer in the Operating System (OS), rather than monitoring at the switches in the network. Mahout does timely management of elephant flows through an in-band signaling mechanism between the shim layer at the end hosts and the network controller. At the switches, any flow not signaled as an elephant is routed using a static load-balancing scheme. Only elephant flows are monitored and managed by the central controller. The combination of end host elephant detection and in-band signaling eliminates the need for per-flow monitoring in the switches, and hence incurs low overhead and requires few switch resources. In Mahout, a shim layer on each end-host monitors the flows originating from that host. When this layer detects an elephant flow, it marks subsequent packets of that flow using an in-band signaling mechanism. The switches in the network are configured to forward these marked packets to the Mahout controller. This simple approach allows the controller to detect elephant flows without any switch CPU and bandwidth-intensive monitoring. The Mahout controller then manages only the elephant flows, to maintain a globally optimal arrangement of them.

At the time when Mahout emerged, Apache Hadoop was the dominant open platform for storing and processing large datasets using the MapReduce paradigm and was initially developed to build indexes for web-scale search engines. Due to the prevalence of Hadoop in industry, as well as research which indicated that a large family of popular ML algorithms can be reformulated under the MapReduce paradigm, Mahout initially focused on MapReduce-based algorithm implementations. These implementations have been widely used by leading web companies including Twitter, LinkedIn and Foursquare and are available in major commercial cloud offerings such as Amazon's Elastic MapReduce service and Microsoft's Azure HDInsight [74].

2.6.1.2. MLlib

In [75], an analysis is made on MLlib's architecture and purpose. MLlib was made to give Spark a distributed ML library. The library targets large-scale learning settings that benefit from data-parallelism or model-parallelism to store and operate on data or models. MLlib consists of fast and scalable implementations of standard learning algorithms for common learning settings including classification, regression, collaborative filtering, clustering, and dimensionality reduction. It also provides a variety of underlying statistics, linear algebra, and optimization primitives.

MLlib's tight integration with Spark results in several benefits. First, since Spark is designed with iterative computation in mind, it enables the development of efficient implementations of large-scale ML algorithms since they are typically iterative in nature. Improvements in low-level components of Spark often translate into performance gains in MLlib, without any direct changes to the library itself. Second, Spark's vibrant open-source community has led to rapid growth and adoption of MLlib. Third, MLlib is one of several high-level libraries built on top of Spark. As part of Spark's ecosystem, MLlib provides developers with a wide range of tools to simplify the development of ML pipelines in practice.

2.6.2. Big Data Analysis Techniques

An investigation was performed by Mostafaeipour et al. [76] on the performance of Hadoop and Spark platforms on ML algorithms. After running the K-means algorithm on datasets with different sizes with both Hadoop and Spark frameworks, the results show that the runtime of the used algorithm implemented on Spark is 4 to 4.5 times faster than Hadoop. Also, Hadoop used more resources like the central processor and the networking of the system. But Spark used more RAM than Hadoop instead.

The comparison [77] concentrates on different ML frameworks for Big Data. The main idea of this comparison is to provide scalable ML tools including Mahout, MLlib, H2O, and SAMOA. Furthermore, the authors evaluated each tool based on different criteria such as: scalability, fault tolerance and usability.

In the paper [78] a Big Data platform was set up, using existing Hadoop ML components like Mahout and Spark MLlib to realize customer automatic reply and information analytic on the field of Electric Power. It established a Big Data platform using ML to process massive electric power customer information and forming an automatic response and redirect mechanism prototype.

In [79], Wan et al. propose and implement a Big Data solution for active preventive maintenance in manufacturing. The architecture consists of four layers: data sources, data transmission, Big Data analysis, and visual presentation. The Hadoop system is used for offline data distribution and calculation, while Apache Storm is employed for real-time processing. The architecture focuses on batch and real-time processing and how to use these components for predictive maintenance.

In [80], Jamal proposes a framework to solve real-time Big Data management, storage, computation challenges and predictive data analytics in Condition Based Maintenance (CBM) systems to predict and monitor changes in components' behavior before they stop. To deal with real-time data, Apache Kafka is used as a distributed messaging system to collect unstructured and semi-structured data. Collected data are delivered to Spark Streaming which is a distributed stream processing engine. MLlib is getting used for data analytics purposes and HDFS as file system.

The authors of [81], presented a model which works on the distributed computing environment to tackle Big Data. The proposed model uses Apache Spark as an execution engine and hive as database. Also, they used HDFS as distributed storage in their hybrid model and Spark MLlib for analytic jobs. The implemented model can handle Big Data efficiently. It can process large datasets and give results in real-time.

As in the case of data processing, the Thesis is based on projects and research using Streaming Data and Batch Data. The analysis of this data will be performed with a combination of Mahout and Spark MLlib frameworks. Finally constructing a system capable of selecting, through data-driven decisions, the appropriate framework for the input analysis.

2.6.3. Mahout vs MLlib

Research [82], developed by testing the usability of the Apache Mahout and Apache Spark MLlib frameworks. According to this research, the range of ML algorithms that support the tools mentioned is quite different from each other. Table 4 shows the algorithms supported by each one during the reported research.

Table 4 – Implemented Algorithms of Mahout and Spark MLlib

Category	Algorithm	Mahout	MLlib
Dimension Reduction	Principle Component Analysis (PCA)	Yes	Yes
	Singular Value Decomposition (SVD)	Yes	Yes
Regression	Linear Regression	No	Yes
	Logistic Regression	No	Yes
Clustering	Hierarchical Clustering	No	Yes
	Distributed-based Clustering	No	Yes
	Centroid-based Clustering (K-means)	Yes	Yes
Classification	Support Vector Machines (SVM)	No	Yes
	Artificial Neural Networks (ANN)	No	
	Decision Tree	No	Yes
	Naive Bayes	Yes	Yes
	Ensemble Methods (Boosting, Random Forest)	Yes (Random forest)	Yes (Boosting and random forest)

In the same research, the authors tested the usability of Apache Mahout, Apache Spark MLlib and Microsoft's CNTK frameworks. According to this research, the range of ML algorithms that support the tools mentioned is quite different from each other. Table 5 shows the algorithms that each supports in relation to the others during the reported research.

Table 5 – Implemented Algorithms of different Frameworks

Category	Algorithm	Apache Mahout	Apache MLlib	Microsoft's CNTK
Dimension Reduction	Principle Component Analysis (PCA)	Yes	Yes	No
	Singular Value Decomposition (SVD)	Yes	Yes	No
Regression	Linear Regression	No	Yes	No
	Logistic Regression	No	Yes	No
Clustering	Hierarchical Clustering	No	Yes	No

Classification	Distributed-based Clustering	No	Yes	No
	Centroid-based Clustering (K-means)	Yes	Yes	No
	Support Vector Machines (SVM)	No	Yes	No
	Artificial Neural Networks (ANN)	No		Yes
	Decision Tree	No	Yes	No
	Naive Bayes	Yes	Yes	No
	Ensemble Methods (Boosting, Random Forest)	Yes (random forest)	Yes (boosting and random forest)	No

In the research experiment [12], the Spark MLlib and Mahout frameworks were compared (using spark_mllib_2.11 and apache-mahout-distribution-0.13.0 versions respectively), based on the K-means, Logistic Regression and Alternating Least Squares algorithms. The experiment was performed by increasing the size of the imported data up to 10 Gigabytes.

The experiments came to the following results:

- MLlib is much faster than Mahout.
- MLlib and Mahout become slower by increasing data.
- For MLlib, when data is extremely large, the memory on Spark is not enough to store newly intermediates results, moreover, MLlib crashes.
- For Mahout, even if it becomes slow when data is large, it is always stable.

Finally, the last research concludes that Mahout is a strict solution in case the user wants to analyze huge amounts of data, but in any other case MLlib performs better, if it has the necessary amount of memory to complete its analysis.

3. Proposed Big Data Management Platform

Following the literature review, this chapter describes the design of the EverAnalyzer platform architecture. In the referenced bibliography, Apache Spark and Apache Hadoop frameworks are compared, as well as the ML libraries using Apache Spark MLlib and Apache Mahout, respectively. Apart from the above comparisons, in which much research has been done, there seems to have been no attempt to manage data analysis using hybrid systems of the two frameworks. EverAnalyzer comes to fill this gap, providing a solution by which an analyst can manage processes using Spark or Hadoop but also the ability of the platform itself to assist the end-users on their decision on the use of the above tools. Having this feature, it makes it easy for any analyst who wants to perform data analysis work on these two (2) frameworks. Ultimately, the platform will be a useful tool for any data analyst that aims to use and export results through Spark and Hadoop.

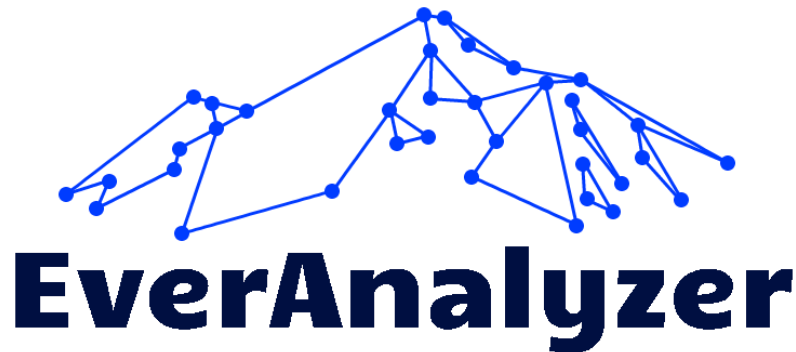


Figure 5 – EverAnalyzer Logo

3.1. Implementation Objectives

Table 6 lists all the objectives implemented in the EverAnalyzer platform.

Table 6 – EverAnalyzer Objectives

Objective's ID	Objective
#1	End-users can log in to the platform with a registration system.
#2	End-users can collect their desired data from the free Social Media Twitter database.
#3	End-users can save their collected data as well as their analyzed data for future use or access.
#4	End-users can do analyzes that they can configure. Within this configuration, there is a choice between using Spark or Hadoop MapReduce and their Mahout and Spark MLlib libraries.
#5	The platform can respond to the optimal use between Spark or Hadoop, depending on the resolution that the end-user wants to perform.
#6	End-users have the ability to pre-process their data.
#7	The results of the end-user analytics will be displayed graphically, in a user-friendly way.

Figures 6-12 show Unified Modelling Language (UML) use case diagrams, aiming to better illustrate the objectives of the platform, in relation to the Users to whom they are addressed.

Figure 6 shows the User interaction with the platform's System, in the phase of authentication processes

- The User can register in the System, but also to log in later.
- The System stores the User's data and authenticates it.

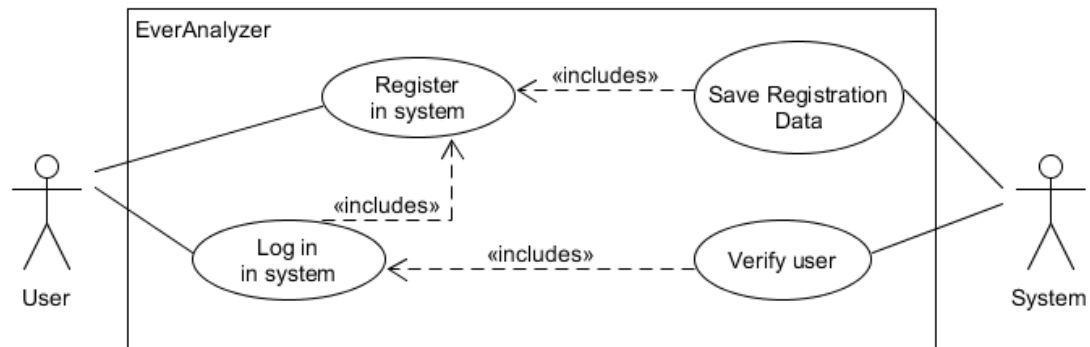


Figure 6 – Use case diagram of Objective #1

Figure 7 shows the operations of the System and the User during the data collection process.

- The User can choose configurations related to the collection of his data.
- The System performs the collection and saves it for later use.

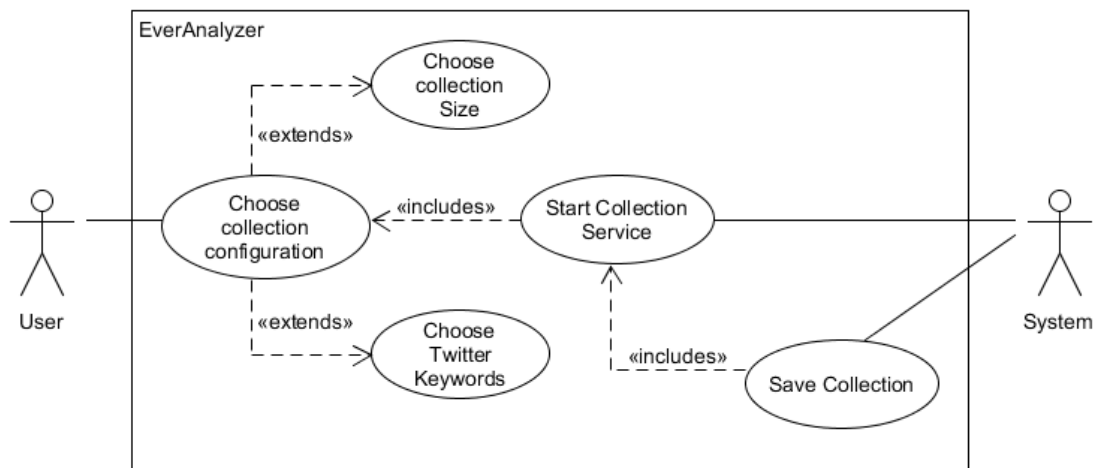


Figure 7 – Use case diagram of Objective #2

Figure 8 shows the data storing process in the platform.

- The User finishes any collecting, pre-processing, processing, or analytic job.
- The System, as responsible execution of the saving process, stores all the useful information of the User's data and the data themselves in its respective storage resources.

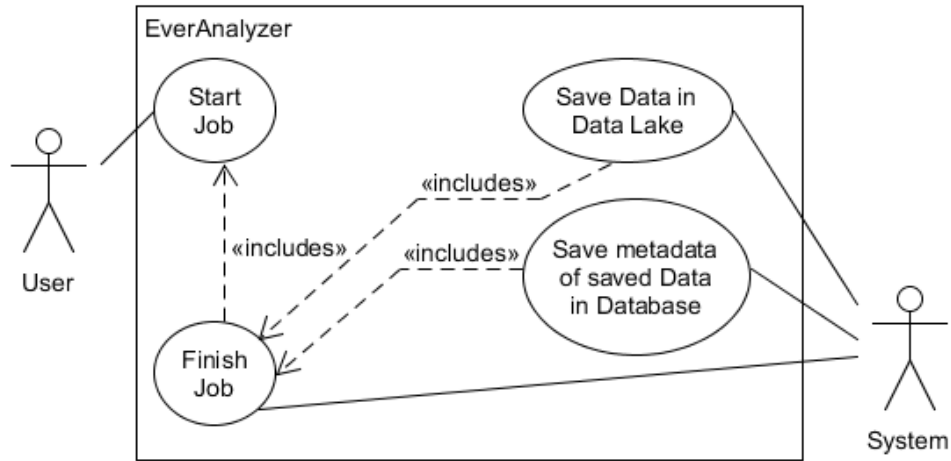


Figure 8 – Use case diagram of Objective #3

Figure 9 shows how the User and the System interact during the construction of the analysis to be performed.

- The User selects a framework to use and then configures the analysis to be executed.
- The System performs the User's Analysis.

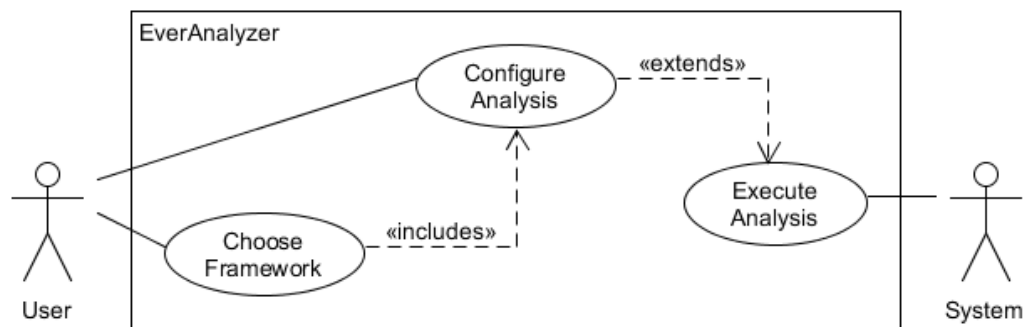


Figure 9 – Use case diagram of Objective #4

Figure 10 shows the framework proposal process, for use in the User's Analysis by the System.

- The User asks the System to give him a proposal and updates the System with the data of the analysis he wants to make.
- The System obtains the User's analysis and after processing it, gives the User the answer of the Framework to use.

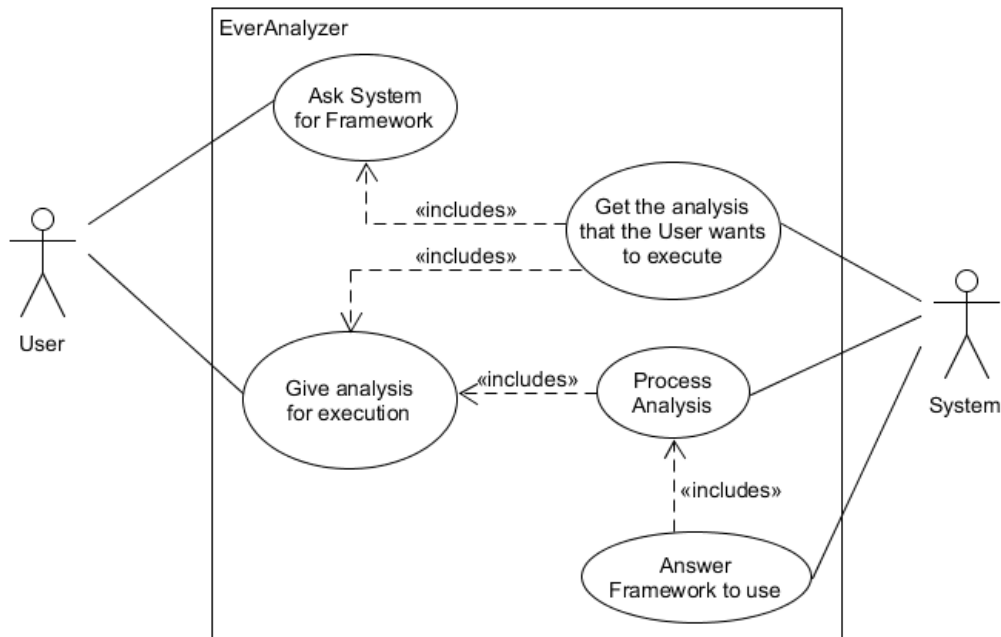


Figure 10 – Use case diagram of Objective #5

Figure 11 shows the data pre-processing process within the EverAnalyzer environment.

- The User configures his pre-processing.
- The System automatically handles the pre-processing of the User performing any configurations given to it.

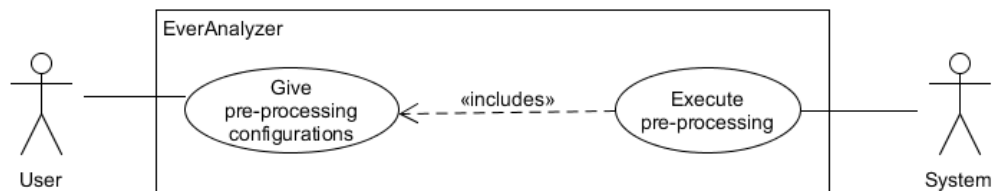


Figure 11 – Use case diagram of Objective #6

Figure 12 shows how the System displays results.

- The User selects the results he wants to see and after the System processes and displays them, can see the final images.
- The System takes the choice of the User for the results to be visualized and produces the corresponding diagrams of the analysis performed.

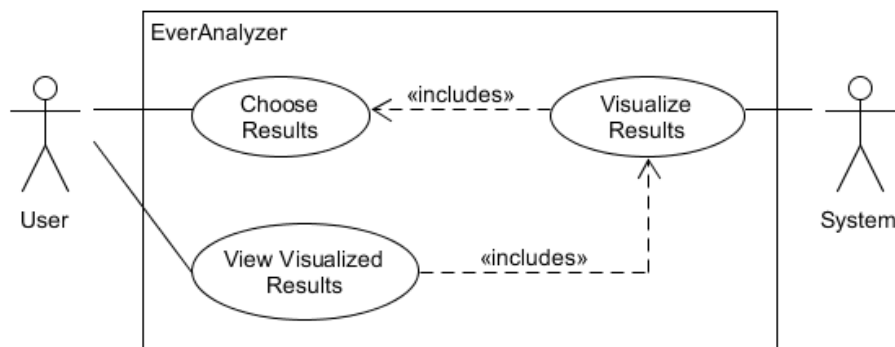


Figure 12 – Use case diagram of Objective #7

3.2. End-users

The developing project is capable of being used by various Users. The following are examples of these and their possible use in the provided platform:

- **Data Analysts:** The Data Analysts who will use the platform, will have the ability to extract results from their data and interpret it.
- **Data Scientists:** The platform provided will allow Data Scientists to experiment with the ways in which a Data Analyst can extract results.
- **Data Engineers:** Data Engineers using the platform will be able to use all the automated processes of the final project as an aid to the rest of their tools.

Figure 13 shows the above examples in a UML format.

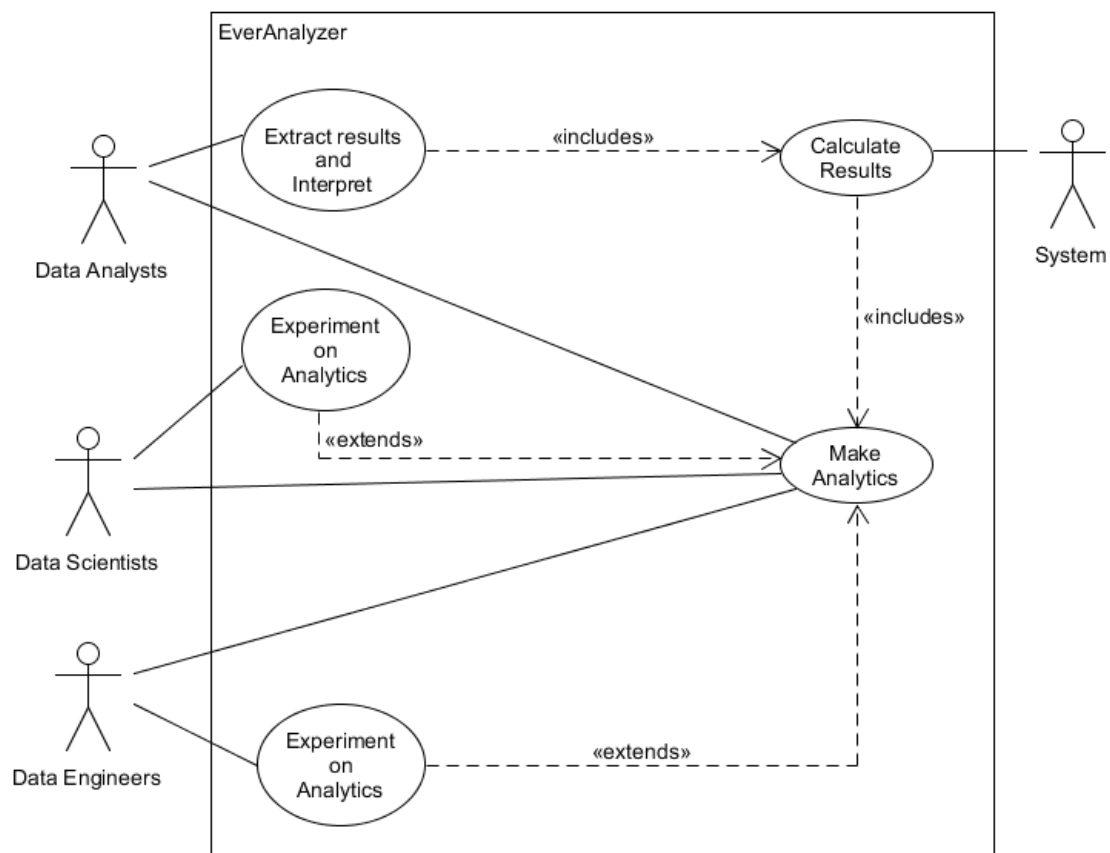


Figure 13 – EverAnalyzer end-users

3.3. Tools and Frameworks

For the construction of the mentioned Platform, tools are used, useful in the field of Big Data, aiming to achieve the final goals that have been set. These tools are as follows:

- **Apache Hadoop MapReduce:** Library for executing Map Reduce algorithms by the platform's Users.
- **Apache Mahout:** Library for the execution of ML algorithms by platform's Users.
- **Apache Spark:** Framework capable of being used to run data analysis algorithms directly on the computer system memory used.
- **Apache Spark MLlib:** Library for running ML algorithms by platform's Users.
- **Apache HDFS:** System for storing files in a distributed manner on a cluster of machines. Used as a Data Lake to store and organize User analytics.
- **Apache Hadoop YARN:** A system responsible for allocating system resources to the jobs that the EverAnalyzer platform is running in the Hadoop cluster and scheduling tasks to be executed on each cluster nodes.
- **MongoDB:** A NoSQL system. This system will be used to store User data, as well as to store the metadata needed to manage the processes that Users want to perform.
- **Twitter API:** Twitter social media system, which allows developers to access a percentage of its database. Used to collect data for the platform's Users.
- **Apache Kafka:** Data collection tool. Used by the Platform to collect Twitter data.
- **Apache Flume:** Tool for transferring data between sources to sinks, using channels. In this Platform the tool takes the data from Kafka and transfers it to HDFS.

3.4. Implementation Plan

3.4.1. High Level Architecture Plan

Figure 14 shows a high-level view of the platform's architecture. As displayed, the mentioned platform contains subsystems which concern the interaction with the User and the management and storage of the data transferred within the platform. These subsystems produce the complete platform and aim to complete the set goals.

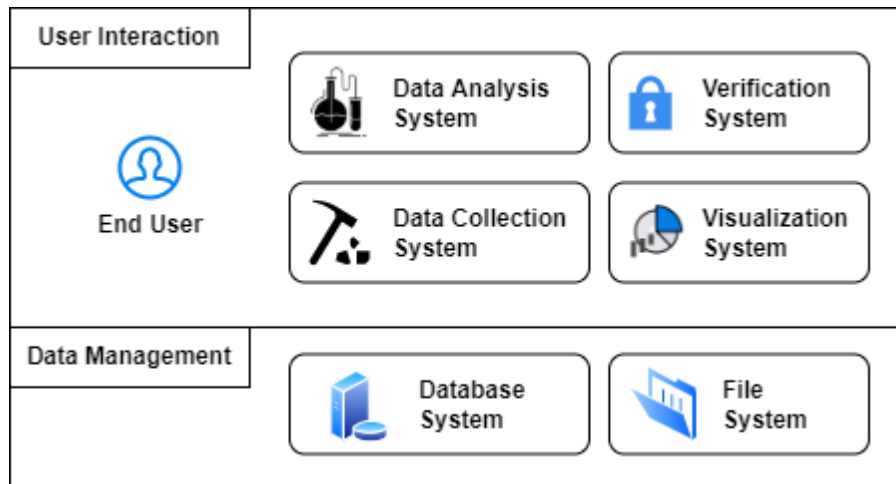


Figure 14 – High Level Architecture

The provided architecture is divided into two (2) layers. These are the Use Interaction Layer and the Data Management Layer. The User Interaction Layer is what contains the subsystems that the user interacts with directly. In contrast to the above, the Data Management Layer is managed primarily by the platform's System. Therefore, the User does not interact directly with the containing subsystems, but indirectly through the data that he provides in the User Interaction Layer.

More specifically all the subsystems of the layers are:

- **Verification System:** In the Verification System, end-users can register and log in to the platform.
- **Data Collection System:** In the Data Collection System, each User will have the ability to give the configurations needed to collect the wanted data.
- **Data Analysis System:** In Data Analysis System, end-users are given the ability to configure analytic jobs that they want to perform and then execute them to get their results.
- **Visualization System:** In the Visualization System, Users receive visual results of their analytic jobs and processes.
- **Database System:** The Database System is used exclusively by the platform itself and not by the Users. It stores useful data for each process and Users of the platform.
- **File System:** The File System is used exclusively by the platform itself and not by the users. It stores, in a distributed way among different machines, all the analyzed, pre-processed, processed and collected data of the platform's Users.

Table 7 shows the correspondence of the platform's Systems in relation to its final objectives.

Table 7 – Correspondence of systems and Objectives

System	Objective ID
Verification System	#1
Data Collection System	#2
Data Analysis System	#4, #5, #6
Visualization System	#7
Database System	#1, #3
File System	#3

3.4.2. Low Level Architecture Plan

Figure 15 shows a low-level view of the platform's Architecture. The figure is displaying the interactions between all the subsystems of the product. All the functionalities of the subsystems are explained in the Sections 3.4.2.2 through 3.4.2.6.

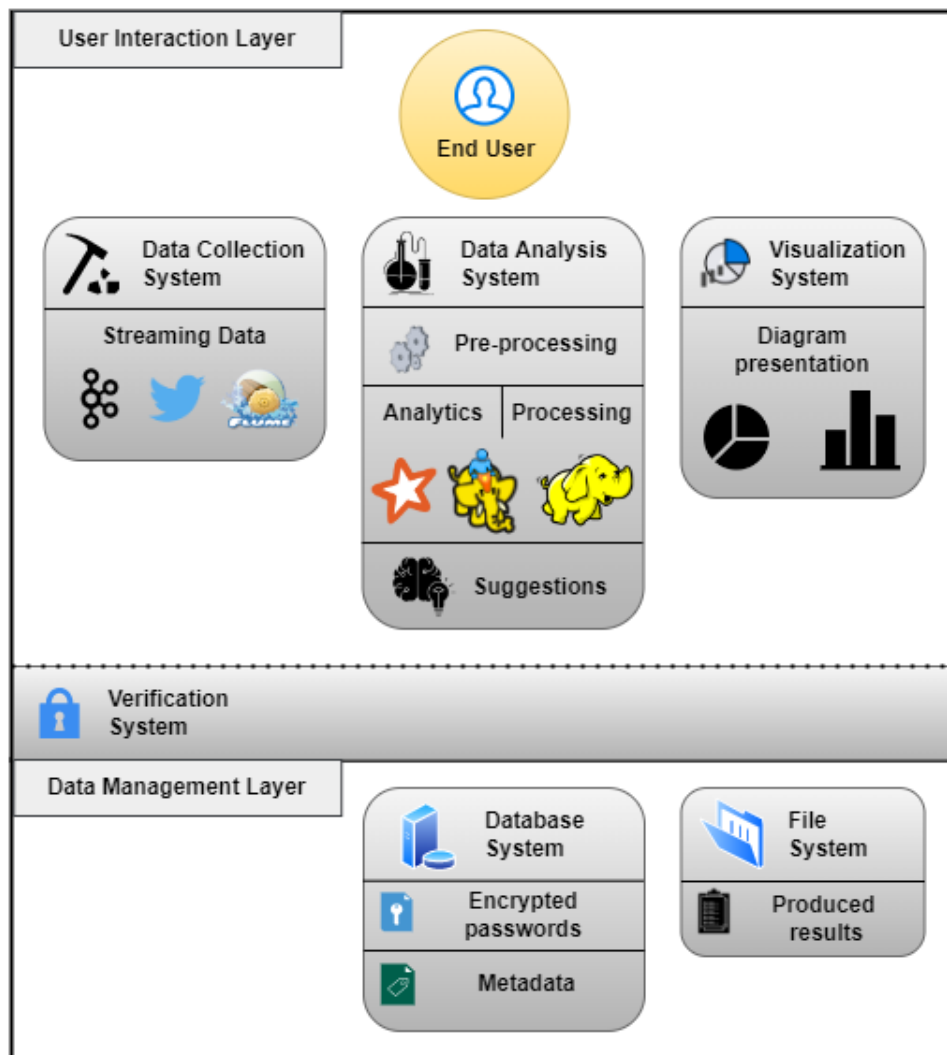


Figure 15 – Low Level Architecture

3.4.2.1. Verification System

The Verification System exists to authenticate different Users in the platform. It supports the following functionalities:

- End-user authentication and login.

- Registration of new Users within the System.
- Gives to the end-users the ability to logout of the System.
- Checks for existing or non-existing User within the System from the browser session where the platform is displayed, to prevent misuse of the tool by non-authenticated end-users.

3.4.2.2. Data Collection System

The Data Collection System gives its end-users the ability to collect data from the Twitter database. It supports the following functionalities:

- Customization of the number of tweets to be collected by the end-user.
- Customization of the words that User wants to search in the Twitter database.
- Able to assign a name (label) to the collected data by the end user.
- Collects data, based on the configurations of the end-user.

3.4.2.3. Data Analysis System

The Data Analysis System is the largest and at the same time the most important part in the platform's architecture. It is the system from which the User can pre-process and analyze his data. Its functionalities are divided into three categories. Below are all the functionalities and their categories:

Pre-processing

- The system can check if there is collected data that the User can pre-process. If there are no collected datasets, then it informs the end-user accordingly.
- If there is collected data, the system presents to the User datasets that it has collected and displays metadata about them (labels, words used for the collection, data size, etc.). These are presented, for the User to choose which dataset to pre-process.
- Once the Users selects the collected dataset for pre-processing, they can configure which tweets they want to retain based on their containing fields.
- After the Users select the collected dataset for pre-processing, they can configure which fields they want to remain in all the tweets in the new pre-processed dataset.
- The User can give a name to the dataset to be produced by the pre-processing (label).

Processing

- The platform checks if there are pre-processed data that the User can go through a Wordcount Processing phase, otherwise it informs the end-user about their need to perform this process.
- If there is pre-processed data, the system presents to the User their datasets and displays metadata about them (labels, words used for the collection, data size, performed pre-processing, etc.). These are presented for the User to choose which dataset to go through the Processing job.
- After the selection of the pre-processed data for processing, User is given the ability to use the Hadoop or Spark libraries to execute the Wordcount process.
- In case the User has not decided which framework to use, then the platform can propose to the User a framework. This proposal is based on the sizes of the machine's System storage and RAM available for use by the platform. In addition to the above, the system will look at old processes, if any, to provide an answer that will be based on the knowledge that the System has gathered.

- The User can give a name to the dataset that will be generated by the Wordcount process (label).

Analytics

- The platform checks if there are pre-processed data that the User can use them to make an analytic job, otherwise it informs him about their need to perform this process.
- If there is pre-processed data, the platform presents to the user their datasets and displays metadata about them (labels, words used for the collection, data size, performed pre-processing, etc.). These are presented for the User to choose which dataset to go through an analysis.
- After the selection of the pre-processed dataset, the User can use the Mahout or Spark MLlib libraries to perform an analysis
- In case the User has not decided on which framework to use, then the platform can propose a framework instead. This proposal is based on the size of RAM and dataset that the user wants to analyze.
- The User can give a name to the dataset that will be generated by the analysis (label).

As described above, the System can make proposals to the Users on what framework they should probably use on given datasets. This proposal is a suggestion from the platform's System that is working differently in processing and analytics proposals. The checks performed by the System, before it makes a proposal, are based on the literature review that has been developed.

- **Processing proposal:** As described in the literature, Spark will be faster from Hadoop's MapReduce on the same use cases, using the same resources, most of the times. As long as the datasets volume is expanding in different use cases, Hadoop's MapReduce will be able to surpass Spark in terms of execution speed, only after a point that the RAM of the distributed cluster will not be enough for Spark. Spark is good to make processing jobs when the dataset is able to fit in the System's RAM. EverAnalyzer is using this exploit to guess the existing point, where after that aforementioned point Hadoop's MapReduce will be superior for the Users. In more detail, for every process done, the platform's System saves the speed of the process and the framework used. For each query of proposition, the below algorithm is triggered:
 - For each past processing (A), using a smaller dataset, exploit Hadoop:
 - Check if there is any existing past processing (B), that used Spark and the dataset was smaller than the processing of A.
 - For each B, if the B processing had slower execution speed than A, then propose the Hadoop framework.
 - For each past processing (A), using a bigger dataset, exploit Hadoop:
 - Check if there is any existing past processing (B), that used Spark and the dataset was bigger than the processing of A.
 - For each B, if the B processing had faster execution speed than A, then propose the Spark framework.
 - If there has not yet been any propose, then give a proposal with 50% chance.
- **Analytics proposal:** The analytics proposal, unlike the Processing proposal, is not that interesting. Based on the literature, Spark MLlib will surpass Mahout always in terms of execution speed, except for the times that the dataset for

analysis is larger than the RAM of the platform's System. Spark MLlib is showed to be able to crash when it is not able to handle the whole datasets inside the Systems RAM, but as well it is showed by the literature to be much slower in these cases. Thus, the algorithm that is getting used for proposition of the framework to be used for the Analytics proposal is only considering the System's RAM. If the System's RAM is smaller than the given dataset, then the proposal is Spark MLlib. In the other hand, if the System's RAM is larger than the given dataset, then the proposal is Mahout.

3.4.2.4. Visualization System

The Visualization System is used to display the analyzed results to the User. Aiming to help interpret the analyzes performed from the Data Analysis System. Below are the systems functionalities:

- The System checks if there are analyzed data that Users can visualize. If not, then the System informs them about their need to perform the visualization.
- In case there are analyzed data, then they appear together with various metadata related to their analysis (labels, words used for the collection, data size, pre-processing performed, analysis applied, etc.).
- When the user selects the analyzed data to visualize, then the corresponding visualization is displayed in the platform.

3.4.2.5. Database System and File System

Data Management Layer (Database System and File System) systems work in conjunction with all the other subsystems of the platform. They exist to store all the exported information of every User and process. But also, the storage of all metadata that ultimately help the better operation of the platform. In addition to the above, these systems give the User the ability to access all the data it has generated and collected.

3.4.3. System Interaction Plan

The UML component diagram presents the subsystems as separate components in interaction with the end-user. This diagram is shown in Figure 16.

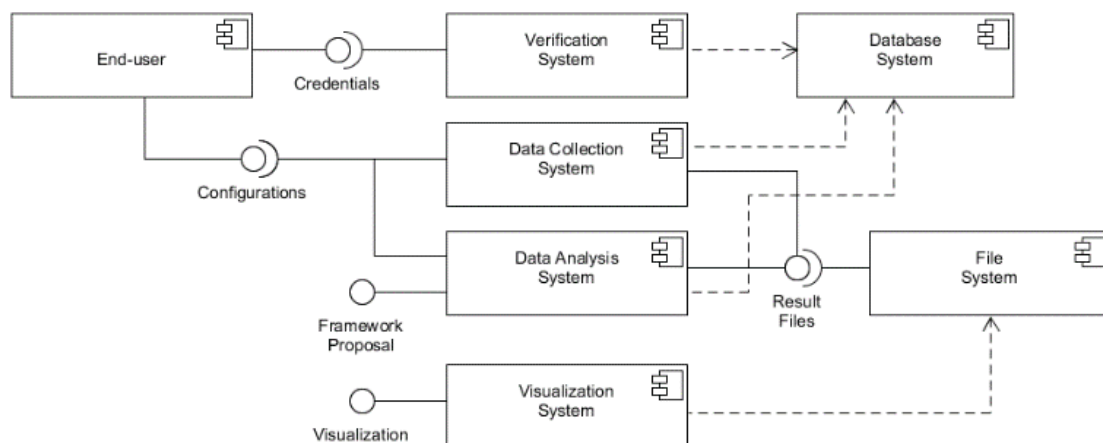


Figure 16 – Component Diagram

More specifically, the end-user provides the data needed to the Verification, Data Collection and Data Analysis Systems to authenticate, as well as gather and perform processes on the needed datasets. The last two systems export results, leaving them

within the File System in an organized manner. In addition to the above, Data Analysis System extracts the response of the framework that the User is suggested to use, between Spark and Hadoop and their respective libraries. The Visualization System generated visualizations of the results of the end-user's analytics.

Database and File Systems are the basis of the other systems, as they depend on them for its proper operation. Verification, Data Collection and Data Analysis Systems store in the Database System information for User authentication and metadata for the procedures performed. While the File System is used by the Visualization System to find the files for visualization.

3.4.4. Execution Plan

In conjunction with the component diagram, a UML class diagram is then used (Figure 17) to clarify the procedures performed by Users when using the platform.

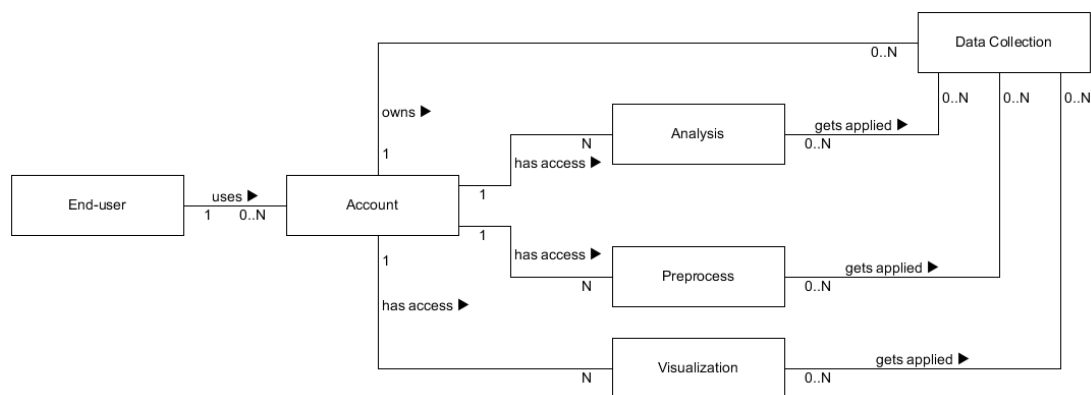


Figure 17 – Class diagram

In the class diagram, the following are displayed:

- End-users can connect to the platform with one or more of their own accounts.
- Each account has access to many data processing, analysis, and visualization algorithms.
- Each account has access to zero or more datasets.
- Many different procedures can be applied to datasets that the User has access to.

This diagram shows to a large extent the Verification, Data Collection, Data Analysis, Visualization Systems, in contrast to the File and Database Systems. This is because the last two as explained in the component diagram are the basis of the rest. Because of them, the other subsystems of the platform can perform their processes.

4. Platform Implementation

4.1. Installation Guide

4.1.1. Prerequisites for Installation

EverAnalyzer can be installed in the Windows OS. Also, the following must be installed in the machine:

- Apache Hadoop v3.1.0
 - HDFS and YARN must be executable
 - There should be a system variable named "HADOOP_HOME" placed directly in the Hadoop directory.
- Apache Flume v1.9.0
 - Flume must be executable.
 - There should be a system variable named "FLUME_HOME" placed directly in the Flume directory.
 - The Flume directory should be mounted directly on the C drive of the user's computer.
 - The Flume directory should be named "apache-flume-1.9.0-bin".
- Apache Kafka v2.13-2.7.0
 - Kafka must be executable.
 - There should be a system variable named "KAFKA_HOME" placed directly in the Kafka directory.
- Apache Mahout v2.13-2.7.0
 - Mahout must be executable.
- Apache Spark v3.1.2
 - Spark must be executable.
- Eclipse IDE for Enterprise Developers (Not Mandatory if met in another way)
- Apache Tomcat v9.0 Server (Not Mandatory if met in another way)
 - Tomcat server must be installed within Eclipse.
 - Tomcat server must be able to run on localhost.

The user who wants to install the platform should have direct access to the C drive of the computer, as the platform will place a new directory inside it, to be able to manage some of its functions. Therefore, if requested by the platform (via a Windows alert message), the user should be able to grant it access.

To facilitate the above installations, a Github repository has been created which contains all the necessary frameworks in the requested versions:

<https://github.com/karamolegkos/EverAnalyzer>

Also, this repository contains the java projects of EverAnalyzer that contain the whole platform.

4.1.2. Installation Process

The EverAnalyzer Github repository contains two (2) projects that need to be downloaded to install the EverAnalyzer platform. These projects are named as:

- **EverAnalyzer:** The main platform.
- **EverAnalyzer-Spark:** An assistant for the main platform.

Anyone can get this repository directly from the Github platform or by using any Git tool for version control systems.

Although there are many ways to add projects within eclipse, it will be mentioned below the most secure one, in order for the platform to be correctly executed.

Step 1

Open Eclipse and make two (2) new Java Dynamic Web Projects:

- File → New → Dynamic Web Project.
- Add names (the names “EverAnalyzer” and “EverAnalyzer-Spark” are used).
- Next → Next → Check “Generate web.xml deployment description” → Finish.

Step 2

Install the libraries needed to execute EverAnalyzer, for each java project:

- Copy all the files inside the below folder of the two projects from the Github directory: <Project_Name>/WebContent/WEB-INF/lib/*
- Paste the copied files exactly at the same paths inside of your Eclipse Projects.

Step 3

Install the executable code to be used by EverAnalyzer, for each java project:

- Copy all the files inside the below folder of the two projects from the Github directory: <Project_Name>/src/*
- Paste the copied files exactly at the same paths inside of your Eclipse Projects.

Step 4

Install the interfaces of EverAnalyzer platform, for each of the two java projects:

- Copy the below folder of the two projects from the Github directory:
- <Project_Name>/WebContent/Interface
- Paste the copied folder with all the contained files exactly at the same paths inside of your Eclipse Projects (The WebContent folders of Eclipse).

Step 5

For each project:

- Right click on the Java Project.
- Build Path → Configure Build Path → Libraries → Add JARS... → <Project_Name> → WebContent → WEB-INF → lib
- Add all the JARS of every lib’s subfolder.
- Click Add JARS until we have added all the needed JARS (Use the Key Shortcut CTRL+SHIFT for the selection of many files in order).

- Click “Apply and Close”

Step 6

For each project:

- Right click on the Java Project.
- Properties → Deployment Assembly → Add → Java Build Path Entries → Next
- Choose all the Java Archives (JARs) that are displayed (Use the Key Shortcut CTRL+A for the selection of all the content)
- Finish → Apply and Close

Step 7

For each project:

- Left click on the Java Project.
- In the Eclipse Toolbar click the “Project” and then “Clean...”
- Remove the “Clean all projects”.
- Choose only the selected project.
- Click “Clean”

Step 8

Make sure that Eclipse has the "Servers" tab available (if not then follow Step 9, otherwise go straight to Step 10).

Step 9

Add the “Servers” tab in Eclipse’s floating information:

- On Eclipse Toolbar Click “Window”.
- Then click “Show view” → Servers.

Step 10

Increase Tomcat’s timeout policy:

- Double click the Tomcat server inside of the tab “Servers”.
- Find the heading “Timeouts”.
- Increase the field “Start (in seconds)”. This field must be increased enough to let EverAnalyzer execute. The platform is using many different JARs making it demanding on resources. It is possible that the server will not be able to start before EverAnalyzer is ready. For slow machines this field must have a value of at least 180 seconds.

Step 11

Add your project int Tomcat server:

- Right click on Tomcat server.
- Click “Add and Remove...”.
- Make sure that only the two projects by EverAnalyzer are deployed in Tomcat. Add the by moving them in the Configured projects in the right of the window.
- Click “Finish”.

Now EverAnalyzer is ready for execution.

4.1.3. Execution Process

Follow the steps bellow to execute EverAnalyzer:

- Open Eclipse's "Servers" tab.
- Select the Tomcat server.
- While Tomcat is selected, press "start server" in the upper right corner of Eclipse's floating information (The green triangle).
- Open any browser and navigate to the below Uniform Resource Locator (URL)

<http://localhost:8080/EverAnalyzer/Interface/Views/Verification/login.jsp>

4.2. Implementation Technologies

EverAnalyzer used the technologies below to be able to provide the platform to its users:

- **Java:** Used as the main programming language of the platform. It is used in every functionality by handling the logic of the platform's System.
- **Java Server Pages (JSP):** Used in cooperation with Java to make the Java Web Application of the platform. Like Java, JSP assisted in some of the EverAnalyzer subsystems as well. This is the main tool that is getting used by the Verification System.
- **Hypertext Markup Language (HTML):** HTML is the most popular language for writing web pages. It assisted EverAnalyzer on all the interfaces by making their structures to be handled by CSS and JS.
- **Cascading Style Sheets (CSS):** Used in making some custom styles on the interfaces of the platform. It helped to give EverAnalyzer a unique taste in its design.
- **Javascript (JS):** This programming language was used in almost all the EverAnalyzer interfaces. It helps to give some logic in the front-end of the platform. This made the Web Application of the platform faster to respond to the user's actions.
- **Bootstrap:** Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It is getting used by EverAnalyzer for user friendly warnings and messages on the front-end side of the platform.
- **jQuery:** The purpose of jQuery is to make it much easier to use JS on a website, while being lightweight as a framework as well. EverAnalyzer used this framework in cooperation with JS for some functionalities of the Web Application.

EverAnalyzer was build and tested on one machine with Windows 10 as its OS, using a total of 8 GB RAM and 237 GB of HDD.

4.3. User Manual

This Section introduces how a User can use EverAnalyzer. Each interface of the platform is presented per sub-section. These sub-sections are divided as follows:

- **Authentication Interface:** The registration and authorization phase of the User.
- **Homepage Interface:** The interface to assist the User in the collection of the needed datasets.
- **Collection Interface:** The interface with which the platform guides the User to collect data from the Twitter API.
- **Pre-processing Interface:** The interface given to the User for the pre-processing of the datasets.
- **Processing Interface:** An interface giving the ability to make Wordcount jobs in the pre-processed datasets.
- **Analytics Interface:** An interface assisting the User to make analytic jobs on pre-processed datasets.
- **Visualization Interface:** An interface with the ability to make visualizations of all the datasets made by the processing and the analytics of the User.
- **Management Interface:** An interface containing all the datasets of the User. Giving the ability to parse and download all the information owned by each User.

4.3.1. Authentication Interface

For the access of the Users in the EverAnalyzer platform, an interface has been created as the access of the Users to their account. The Verification System is installed with it and they interact harmoniously to secure the information of the EverAnalyzer Users. Figures 18 and 19, show the sign-in and sign-up interfaces. More specifically, the User can use the sign-up page, giving a password and a username to use for his authentication in the sign-in page of the platform. The password of the User is getting encrypted before its saving in the Database System.

In both interfaces the platform will make user-friendly warnings in case that he is doing something wrong, like giving different values in the “Password” and the “Verify password” fields or trying to sign-up with an already existing username (Figures 20 and 21).



Use your credentials to Log in.

User name

Password

[or Sign up...](#)

Sign in

Figure 18 – Sign-up interface



Fill in the form to register.

User name

Password

Verify Password

[or Sign in...](#)

Sign up

Figure 19 – Sign-in interface



Fill in the form to register.

User name

Password

Verify Password

[or Sign in...](#) **Sign up**

This user already exists!

All fields must be filled!

Figure 20 – Existing User / Sign-up



Use your credentials to Log in.

User name

Password

[or Sign up...](#) **Sign in**

The username is not matching with the password!

Figure 21 – Wrong Credentials / Sign-in

4.3.2. Homepage Interface

After the User logs in the platform, a side bar appears, so that the User has the ability to navigate between the functionalities of EverAnalyzer and use them. These functionalities consist of the Collection, Pre-processing, Processing, Analytics, Visualization and Management, which will be thoroughly described below. Giving the user the ability to use all the subsystems described in the chapters above. In addition, the logo of the platform is displayed. Also, the user is given the ability to log-out from EverAnalyzer. Figure 22 shows the Homepage interface.

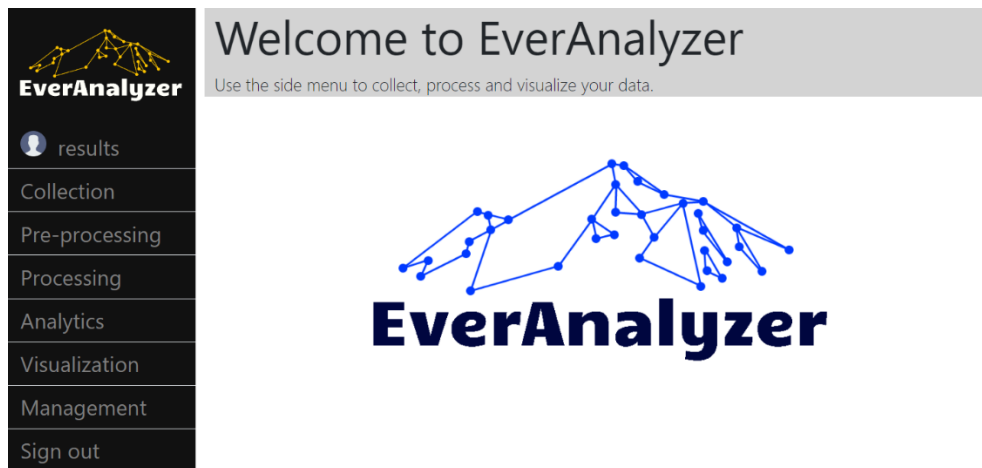



Figure 22 – Homepage Interface

4.3.3. Collection Interface

The Collection Interface gives the User the ability to make collection datasets from the Twitter API. The User can provide the label of the collection as well as the number of Tweets to gather and the keywords to use. These keywords are essential for the gathering, making Twitter API return Tweets containing the keywords that the User has given. Figure 23 shows the Collection Interface. The interface provided to the User, will assist him in avoiding making mistakes like, typing a label that already exists in one of his current labels of his datasets, or not filling a field. This is happening with the use of user-friendly warnings in the moment that the User is making something wrong, while preventing to let the User submit the collection without a valid collection form.



[results](#)
[Collection](#)
[Pre-processing](#)
[Processing](#)
[Analytics](#)
[Visualization](#)
[Management](#)
[Sign out](#)

Collection

Collect data using Twitter API.

Give a label for your collection

Give the amount of Tweets to collect

Give below all the keywords to search for (use commas to separate keywords)

Input your Twitter Consumer Key

Input your Twitter Consumer Secret

Input your Twitter Token


Input your Twitter Secret

Collect Data

Figure 23 – Collection Interface

4.3.4. Pre-processing Interface

The Pre-processing Interface provides all the User's collection datasets as choices for pre-processing as shown in Figure 24. In more detail, a metadata summary is displayed for each collection dataset. This metadata consists of the label of the dataset, the date it was collected, the number of Tweets contained in it, its size in bytes and the words used to collect the dataset from Twitter.



[results](#)
[Collection](#)
[Pre-processing](#)
[Processing](#)
[Analytics](#)
[Visualization](#)
[Management](#)
[Sign out](#)

Pre-processing

Pre-process data using EverAnalyzer.

Label: **anaemia** Date: **2022/06/11**

Tweets amount: **500** Size: **689352 bytes**

Words: **anaemia children haemoglobin fatigue weakness dizziness**

Select Dataset

Label: **cancer** Date: **2022/06/11**

Tweets amount: **500** Size: **643186 bytes**

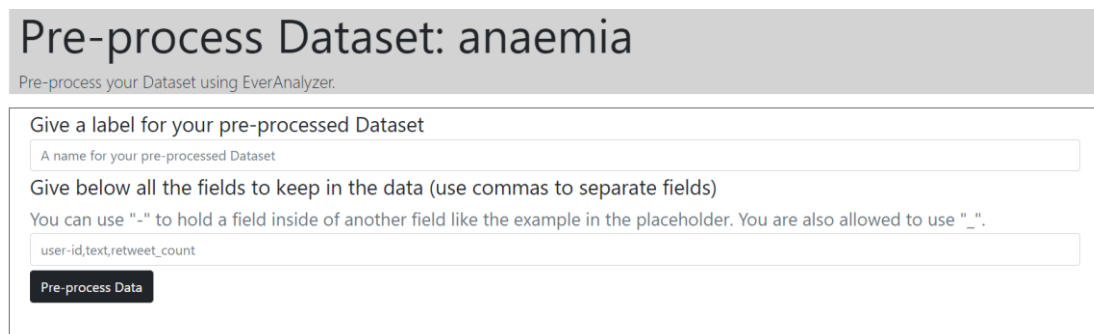
Words: **cancer organ death tumour prostate breast**

Select Dataset

Figure 24 – Collection datasets

When the User decides which collection dataset to pre-process, then by pressing the button "Select Dataset" the platform provides the pre-processing form. In this form the User can choose which fields of the raw Tweets should be kept. This choice is not bound by the JavaScript Object Notation's (JSON) schema, because of the ability of the platform to detect all the fields that the User requests, no matter how nested they are within the JSON schema. After the pre-processing only the Tweets containing the

needed fields are still existing in the dataset having only the requested fields. Figure 25 shows the pre-processing form.



The form is titled "Pre-process Dataset: anaemia" with a subtitle "Pre-process your Dataset using EverAnalyzer." It contains a text input field for a label, instructions on how to use commas and underscores, a text input field with the example "user-id,text,retweet_count", and a "Pre-process Data" button.

Pre-process Dataset: anaemia

Pre-process your Dataset using EverAnalyzer.

Give a label for your pre-processed Dataset

A name for your pre-processed Dataset

Give below all the fields to keep in the data (use commas to separate fields)

You can use "-" to hold a field inside of another field like the example in the placeholder. You are also allowed to use "_".

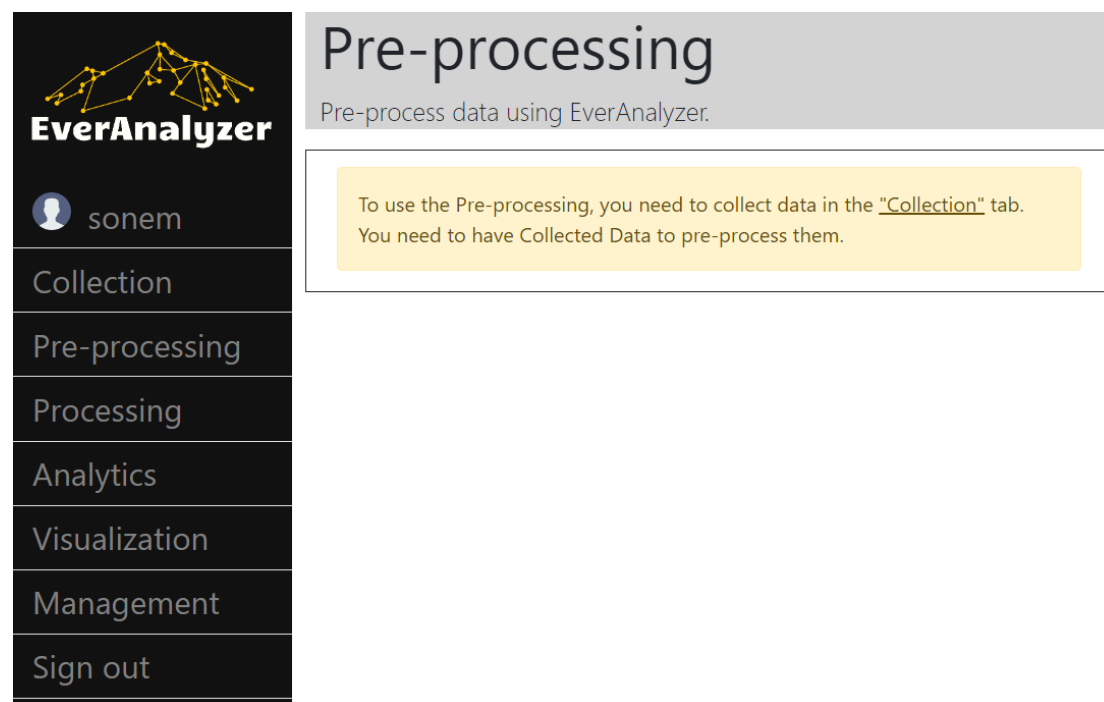
user-id,text,retweet_count

Pre-process Data

Figure 25 – Pre-processing form

The interface provided to the User, will assist him in avoiding making mistakes like, typing a label that already exists in one of his current labels of his datasets, or not filling a field. This is happening with the use of user-friendly warnings in the moment that the User is making something wrong, while preventing to let the User submit the pre-processing without a valid pre-processing form.

If there are not yet any collection datasets, then the interface informs the User for the need to make a pre-processing job. Figure 26 shows this case.



The interface shows a sidebar with the EverAnalyzer logo and a user profile "sonem". The main content area is titled "Pre-processing" with the subtitle "Pre-process data using EverAnalyzer." A yellow warning box states: "To use the Pre-processing, you need to collect data in the 'Collection' tab. You need to have Collected Data to pre-process them." The sidebar menu includes: Collection, Pre-processing, Processing, Analytics, Visualization, Management, and Sign out.

EverAnalyzer

sonem

Collection

Pre-processing

Processing

Analytics

Visualization

Management

Sign out

Pre-processing

Pre-process data using EverAnalyzer.

To use the Pre-processing, you need to collect data in the "Collection" tab. You need to have Collected Data to pre-process them.

Figure 26 – No collection datasets

4.3.5. Processing Interface


The Processing Interface provides all the User's pre-processed datasets as choices for processing as shown in Figure 27. In more detail, a metadata summary is displayed for each pre-processed dataset. This metadata consists of the label of the dataset, the date it was constructed, the number of Tweets contained in it, after the pre-processing, its size in bytes, the pre-processed fields that the user chose to keep in his dataset, as well as the label from the collection dataset that was pre-processed and the words used to collect the dataset from Twitter.

The screenshot shows the 'Processing' interface of EverAnalyzer. On the left is a sidebar with navigation links: results, Collection, Pre-processing, Processing, Analytics, Visualization, Management, and Sign out. The main area is titled 'Processing' with the subtitle 'Process data using EverAnalyzer.' It displays two datasets for selection, each with a 'Select Dataset' button.

Dataset Label	Date	Pre-process Tweets amount	Pre-process Size	Pre-processing Fields	Collection Label	Collection Words
p-anaemia	2022/06/11	285	86873 bytes	retweeted_status-extended_tweet-full_text	anaemia	anaemia, children, haemoglobin, fatigue, weakness, dizziness
p-cancer	2022/06/11	294	90810 bytes	retweeted_status-extended_tweet-full_text	cancer	cancer, organ, death, tumour, prostate, breast

Figure 27 – Pre-processing datasets - processing

When the User decides which pre-processed dataset to process, then by pressing the button "Select Dataset" the platform provides the processing form. In this form the User can make a Wordcount job on the chosen pre-processed dataset. This job can be done using the Hadoop MapReduce or the Spark frameworks. After the processing, the Wordcount job is ready for visualization in the Visualization Interface. This form contains a button named "Suggest" making the platform propose a framework, between Hadoop and Spark, to the User for the requested processing job. Figure 28 shows the processing form and Figure 29 shows the proposal of the platform for the displayed dataset ("p-anaemia"), after the "Suggest" button was pressed.



EverAnalyzer

- results
- Collection
- Pre-processing
- Processing
- Analytics
- Visualization
- Management
- Sign out

Process Dataset: p-anaemia

Process your Dataset using EverAnalyzer. The process will be a word count on the keywords of your Dataset.

Give a label for your Process results


Select below between Hadoop or Spark frameworks

Process

Give me a suggestion based on what I should use

Suggest

Figure 28 – Processing form



EverAnalyzer

- results
- Collection
- Pre-processing
- Processing
- Analytics
- Visualization
- Management
- Sign out

Process Dataset: p-anaemia

Process your Dataset using EverAnalyzer. The process will be a word count on the keywords of your Dataset.

Give a label for your Process results

Select below between Hadoop or Spark frameworks

Process

Give me a suggestion based on what I should use

Suggest

You should use Spark to do your Processing job for the Dataset with the label: p-anaemia

Figure 29 – Processing proposal

The interface provided to the User, will assist him in avoiding making mistakes like, typing a label that already exists in one of his current labels of his datasets, or not filling a field. This is happening with the use of user-friendly warnings in the moment that the User is making something wrong, while preventing to let the User submit the processing without a valid processing form.

If there are not yet any pre-processed datasets, then the interface informs the User for their need to make a processing job. Figure 30 shows this case.

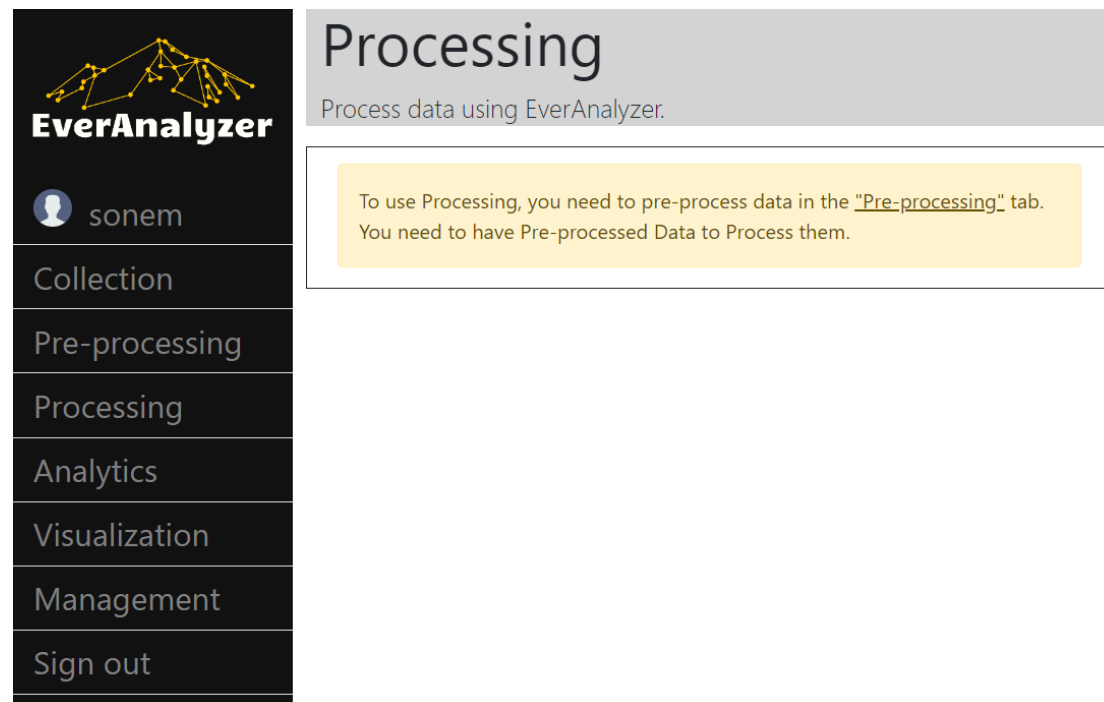


Figure 30 – No pre-processed datasets – processing

4.3.6. Analytics Interface

The Analytics Interface provides all the User's pre-processed datasets as choices for analytic jobs as shown in Figure 31. In more detail, a metadata summary is displayed for each pre-processed dataset. This metadata consists of the label of the dataset, the date it was constructed, the number of Tweets contained in it, after the pre-processing, its size in bytes, the pre-processed fields that the user chose to keep in his dataset, as well as the label from the collection dataset that was pre-processed and the words used to collect the dataset from Twitter.

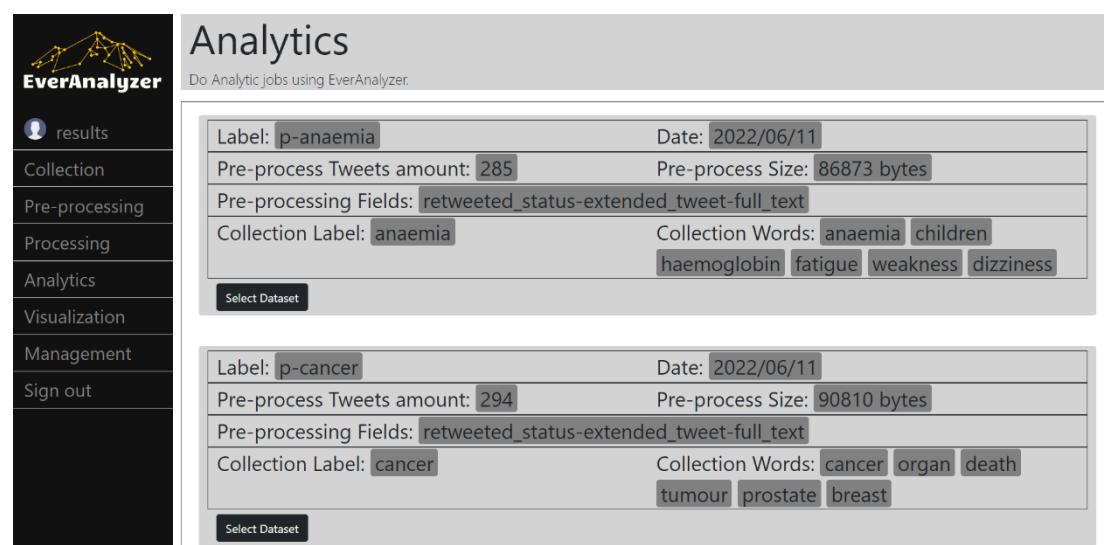
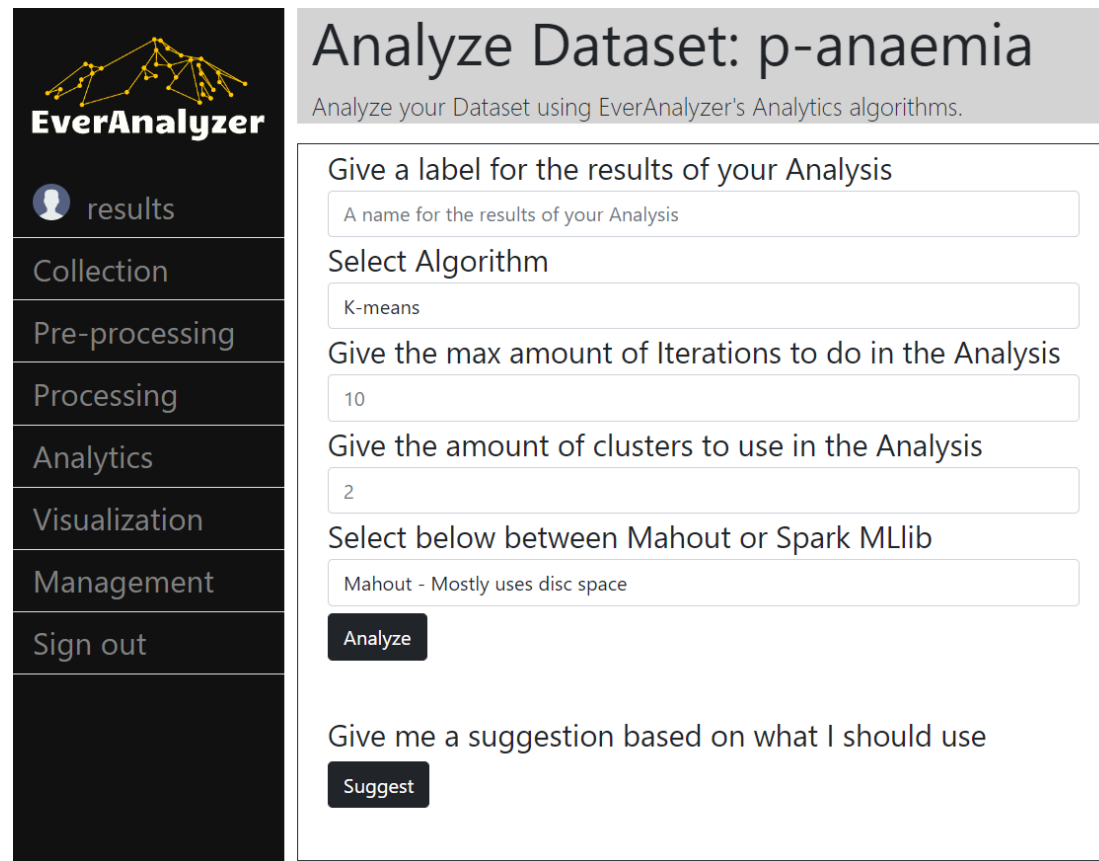



Figure 31 – Pre-processing datasets - analytics

When the User decides which pre-processed dataset to analyze, then by pressing the button “Select Dataset” the platform provides the analysis form. In this form the User can choose an algorithm to use on the chosen pre-processed dataset, containing algorithms like K-means clustering. This job can be done using the Mahout or the Spark MLlib libraries. After the analysis, the job is ready for visualization in the Visualization Interface. This form contains a button named “Suggest” making the platform propose a framework, between Mahout and Spark MLlib, to the User for the requested analytics job. Figure 32 shows the analysis form and Figure 33 shows the proposal of the platform for the displayed dataset (“p-anaemia”), after the “Suggest” button was pressed.





Analyze Dataset: p-anaemia

Analyze your Dataset using EverAnalyzer's Analytics algorithms.

results

Collection

Pre-processing

Processing

Analytics

Visualization

Management

Sign out

Give a label for the results of your Analysis

A name for the results of your Analysis

Select Algorithm

K-means

Give the max amount of Iterations to do in the Analysis

10

Give the amount of clusters to use in the Analysis

2

Select below between Mahout or Spark MLlib


Mahout - Mostly uses disc space

Analyze

Give me a suggestion based on what I should use

Suggest

Figure 32 – Analysis form



results

Collection

Pre-processing

Processing

Analytics

Visualization

Management

Sign out

Analyze Dataset: p-anaemia

Analyze your Dataset using EverAnalyzer's Analytics algorithms.

Give a label for the results of your Analysis

A name for the results of your Analysis

Select Algorithm

K-means

Give the max amount of Iterations to do in the Analysis

10

Give the amount of clusters to use in the Analysis

2

Select below between Mahout or Spark MLlib

Spark MLlib - Mostly uses RAM

Analyze

Give me a suggestion based on what I should use


Suggest

You should use [Spark MLlib](#) to do your Analytic job for the Dataset with the label: p-anaemia

Figure 33 – Analysis proposal

The interface provided to the User, will assist him in avoiding making mistakes like, typing a label that already exists in one of his current labels of his datasets, or not filling a field. This is happening with the use of user-friendly warnings in the moment that the User is making something wrong, while preventing to let the User submit the analysis without a valid analysis form.

If there are not yet any pre-processed datasets, then the interface informs the User for their need to make a processing job. Figure 34 shows this case.



sonem

Collection

Pre-processing

Processing

Analytics

Visualization

Management

Sign out

Analytics

Do Analytic jobs using EverAnalyzer.

To make analytics, you need to pre-process data in the "[Pre-process](#)" tab.
You need to have Pre-processed Data to Analyze them.

Figure 34 – No pre-processed datasets - analytics

4.3.7. Visualization Interface

The Visualization Interface provides all the User's processed and analyzed results as choices for visualization. The datasets are organized in lists that the User can open and close to view only the processed or the analyzed results as shown in Figures 35 and 36. Like the other interfaces, the lists will let the User know if there are no results in the lists.

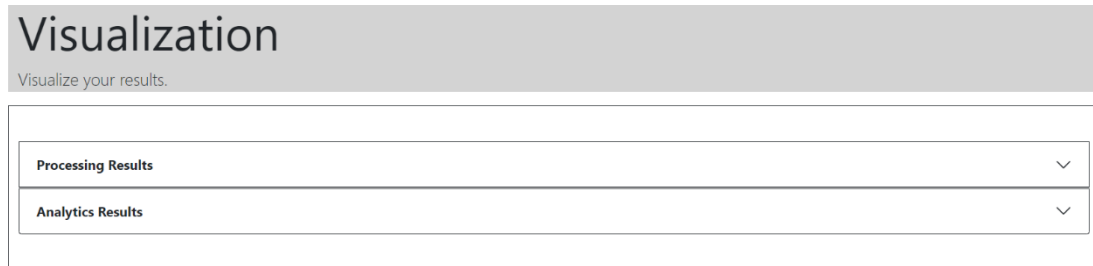


Figure 35 – Visualization lists

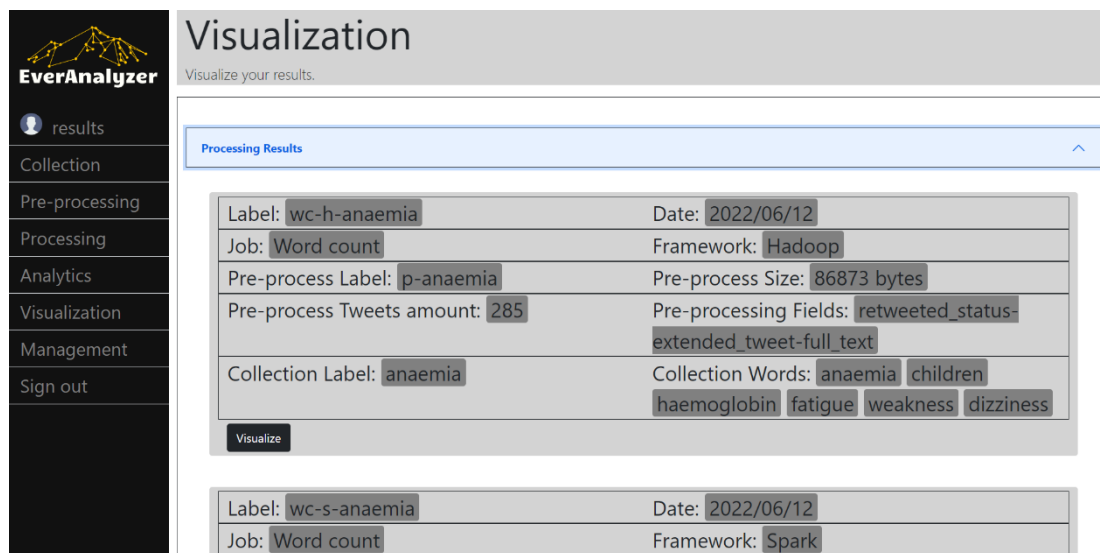


Figure 36 – Visualizable results

By pressing the “Visualize” button on any of the given results the User can get visualizations according to the job that made the chosen result. These visualizations differ on each result, due to the different utility of each visualization method for each one. Two examples are provided in the Figures 37 and 38.

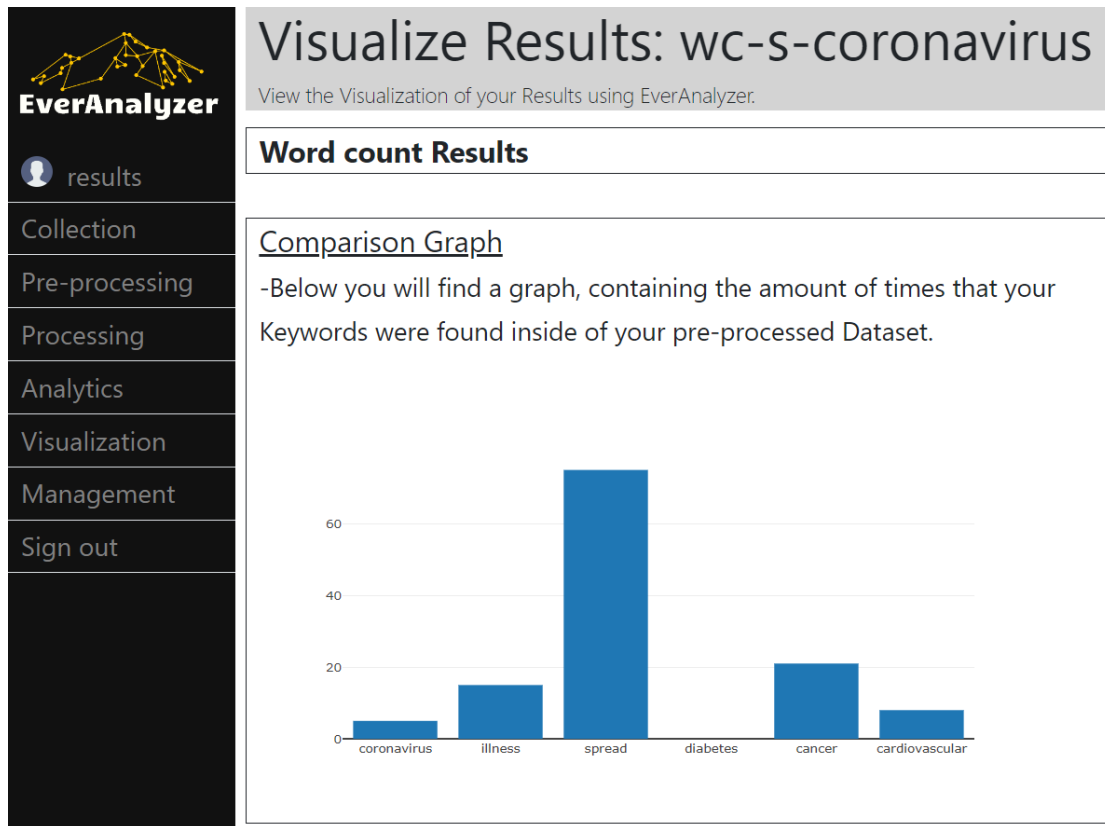


Figure 37 – Wordcount visualization

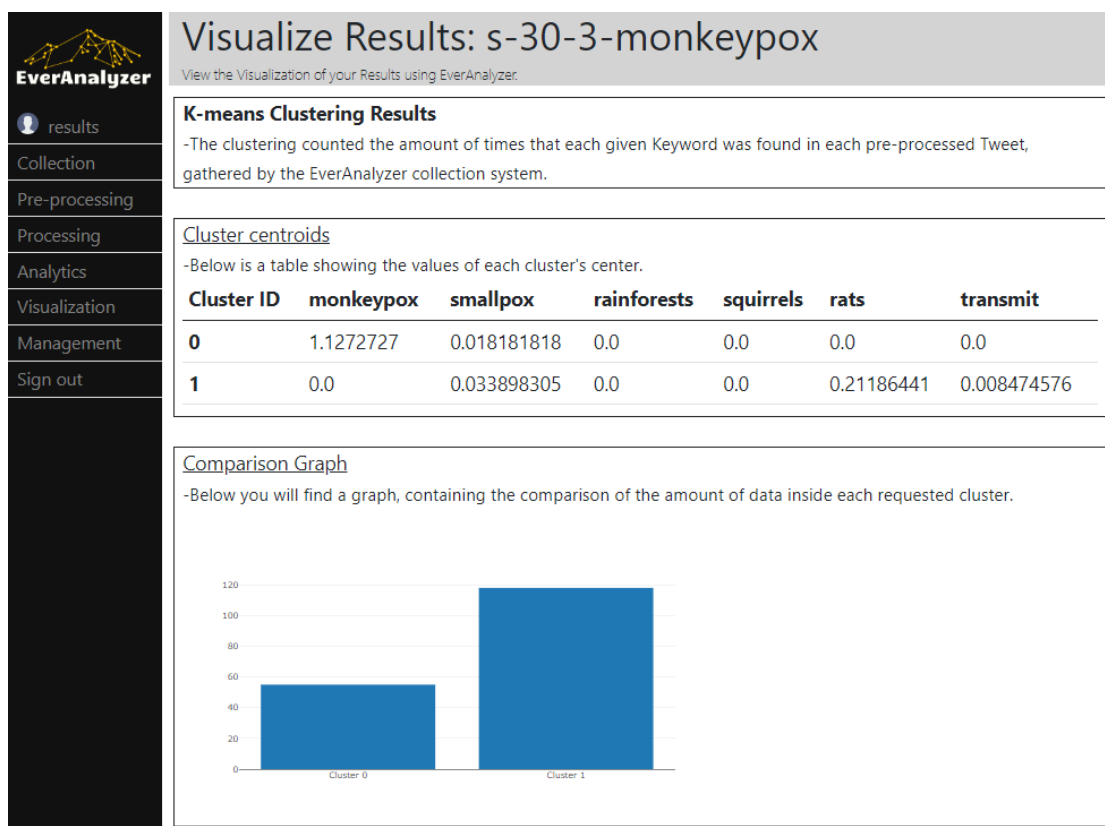


Figure 38 – K-means visualization

4.3.8. Management Interface

The Management Interface allows the EverAnalyzer Users to view all their gathered, pre-processed, processed and analyzed datasets and results exactly as formulated in their saved files. It gives them the ability to view the given datasets and results as raw information but download all the information from the jobs in their system as well.

For the collected and pre-processed datasets an extra functionality is given, making able the Users to view one by one all the JSON objects inside of their datasets.

The Interface provides all the User's datasets and results as choices for viewing, organized in lists that the User can open and close to view only one category of their datasets as shown in Figures 39 and 40. Like the other interfaces, the lists will let the User know if there are no datasets or results in the lists.

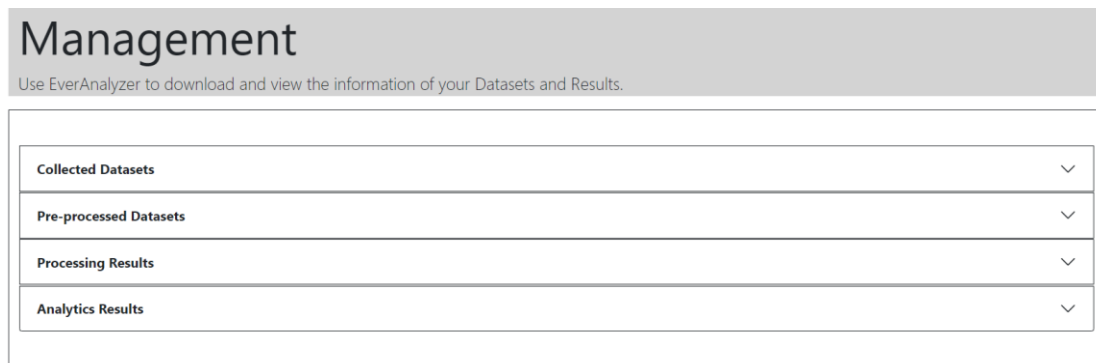


Figure 39 – Management lists

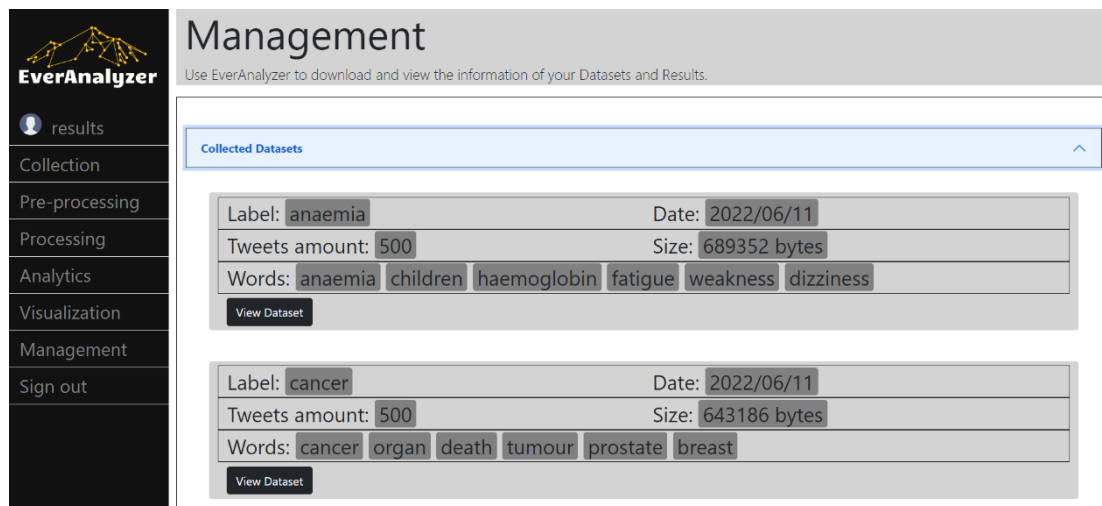
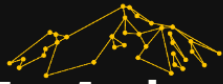


Figure 40 – Management results

Figures 41 and 42 show some examples of how the User is Viewing the datasets and results of EverAnalyzer.



EverAnalyzer

- results
- Collection
- Pre-processing
- Processing
- Analytics
- Visualization
- Management
- Sign out

Viewing Dataset: p-anaemia

View the information of your Dataset.

Pre-process Information

View your pre-processing below


Download Information

Download Information

[next](#)

```
{
  "retweeted_status-extended_tweet-full_text": "Targetted harassment against Muslim activists"
}
```

Figure 41 – Viewing pre-processing



EverAnalyzer

- results
- Collection
- Pre-processing
- Processing
- Analytics
- Visualization
- Management
- Sign out

Viewing Results: wc-h-anaemia

View the information of your Results.

Word count Information

View your Word count results below

Download Information

Download Information

Map-Reduce Results:

children	220
fatigue	2

Figure 42 – Viewing Wordcount

5. Platform Evaluation

5.1. Data Description

Experimentation was done using the EverAnalyzer platform with the proposal mechanism of the Apache Hadoop and Apache Spark frameworks. Systems that use a combination of the two (2) frameworks for data analysis, despite the comparison of Hadoop and Spark, do not seem to have been built in large numbers, so EverAnalyzer is one of the first to try such an approach.

The experiment that took place was purely about the proposals between Hadoop and Spark despite their libraries, namely Mahout and Spark MLlib. According to the literature developed, there is no reason to experiment on Mahout and Spark MLlib as it is clear that Mahout should only be used instead of Spark MLlib if the analysis dataset is larger than the RAM of the given system.

On the other hand, the literature comparing Hadoop and Spark as frameworks reported that there is a point in the size of datasets that Hadoop and Spark will do equal performance, but after it Hadoop begins to prevail. EverAnalyzer uses metadata which it collects after each processing of every User, to better suggest the framework to use.

Health data were used to test this functionality. More specifically, 30 different diseases and conditions were selected, and for each of them 500 Tweets were collected as Streaming Data, through EverAnalyzer, which were then placed within the EverAnalyzer Data Management Layer as Batch Data. For the collection, the experimentation used keywords referred by the World Health Organization to describe each of the diseases and conditions on their webpage [83].

The diseases and conditions used are shown in Table 8.

Table 8 – Diseases and conditions used

Anaemia	Cancer	Cholera	Coronavirus	Influenza	Monkeypox
Obesity	Pneumonia	Smallpox	Syphilis	Tetanus	Yellow fever
Zika virus	Trachoma	Diabetes	Diarrhoea	Ebola virus	Epilepsy
Hepatitis	HIV-AIDS	Depression	Disability	Cardiovascular	Chagas
Dementia	Dracunculiasis	Echinococcosis	Foodborne	Hypertension	Infertility

5.2. Experimentation Environment

The above experiment was performed on a system with 237 Gigabytes HDD and 8 Gigabytes RAM. Also, there was no distribution as there was one (1) cluster node using the distribution of the platform. The machine that run the platform was using the Windows 10 OS. The tools that were exploited used were Apache Hadoop v3.1.0, Apache Flume v1.9.0, Apache Kafka v2.13-2.7.0, Apache Mahout v2.13-2.7.0, Apache Spark v3.1.2, Apache Tomcat Server v9.0, and Twitter API v1.1.

5.3. Experimental Results

To begin with, all the Tweets got pre-processed by EverAnalyzer, by keeping only their fields related to the full text of each one. Hence, the fields that were kept for each retrieved Tweet were the retweeted_status/extended_tweet/full_text. This path was specified as “retweeted_status-extended_tweet-full_text” in EverAnalyzer, showing the path of the attributes inside each JSON Tweet by separating it using the “.” character. The Tweets that did not contain full text did not proceed to the next phase of the experimentation. The final numbers of Tweets that was retrieved and kept for further analysis for each disease/condition are depicted in Table 9.

Table 9 – Pre-processing Results

Disease/Condition (Twitter keyword)	Byte size (Before pre-processing)	Byte size (After pre-processing)	Final number of Tweets (After pre-processing)
Anaemia	689352	86873	285
Cancer	643186	90810	294
Cholera	667851	75305	246
Coronavirus	682377	67983	225
Influenza	635255	80703	268
Monkeypox	52702	48319	173
Obesity	714737	87469	285
Pneumonia	625063	81848	264
Smallpox	659805	92679	299
Syphilis	158457	86303	268
Tetanus	415683	75669	246
Yellow fever	82814	60156	203
Zika virus	673913	92458	279
Trachoma	294028	63508	205
Diabetes	659150	49383	165
Diarrhoea	679323	96373	314
Ebola virus	653989	75345	241
Epilepsy	572757	53907	173
Hepatitis	612424	84098	275
HIV-AIDS	167690	77874	247
Depression	721110	70687	212
Disability	716667	69673	218
Cardiovascular	700503	87562	294
Chagas	624708	74773	251
Dementia	603855	56667	185
Dracunculiasis	119143	78416	255
Echinococcosis	617787	67816	224
Foodborne	163814	84306	264
Hypertension	669548	68421	224
Infertility	322207	49324	155

In sequel, in each pre-processed dataset, a Wordcount job was done regarding the keywords that each dataset had from its collection. The process of this phase was performed for each dataset with the following three (3) steps:

1. Ask for Framework Suggestion
2. Execute the Wordcount with Hadoop
3. Execute the Wordcount with Spark

To this end, it is important to note that the more datasets EverAnalyzer has, the better its suggestions will be, because of the larger amount of knowledge it will contain. Also, for every new dataset after the first one, EverAnalyzer was gathering two new datasets as knowledge, because of the steps (2) and (3) above.

Hence, after the pre-processing of all the data, for each pre-processed dataset, the experimenter asked first the proposition of EverAnalyzer on which framework (Hadoop or Spark) should be used for the given dataset. After getting the response from the platform, the experimenter executed two (2) Wordcount jobs for the current dataset.

By first asking EverAnalyzer for the proposition and then making the execution, the experimenter was able to write down the proposal of EverAnalyzer and check if the proposal was the best of the two (2) for each dataset based on the execution speed. All the captured results are depicted in Table 10.

Table 10 – Processing Results

Disease/Condition	Consecutive Correct Answers	Suggestion Result	EverAnalyzer Recommendation	Expected Recommendation
Anaemia	1	Correct	Hadoop	Hadoop
Cancer	0	Wrong	Hadoop	Spark
Cholera	1	Correct	Spark	Spark
Coronavirus	2	Correct	Spark	Spark
Influenza	3	Correct	Spark	Spark
Monkeypox	4	Correct	Spark	Spark
Obesity	0	Wrong	Hadoop	Spark
Pneumonia	1	Correct	Spark	Spark
Smallpox	0	Wrong	Hadoop	Spark
Syphilis	1	Correct	Spark	Spark
Tetanus	2	Correct	Spark	Spark
Yellow fever	3	Correct	Spark	Spark
Zika virus	0	Wrong	Hadoop	Spark
Trachoma	1	Correct	Spark	Spark
Diabetes	2	Correct	Spark	Spark
Diarrhoea	0	Wrong	Hadoop	Spark
Ebola virus	1	Correct	Spark	Spark
Epilepsy	2	Correct	Spark	Spark
Hepatitis	3	Correct	Spark	Spark
HIV-AIDS	4	Correct	Spark	Spark
Depression	5	Correct	Spark	Spark
Disability	6	Correct	Spark	Spark
Cardiovascular	0	Wrong	Hadoop	Spark
Chagas	1	Correct	Spark	Spark
Dementia	2	Correct	Spark	Spark
Dracunculiasis	3	Correct	Spark	Spark
Echinococcosis	4	Correct	Spark	Spark
Foodborne	5	Correct	Spark	Spark
Hypertension	6	Correct	Spark	Spark
Infertility	7	Correct	Spark	Spark

More specifically, the experiment was able to gather the below results from Table 10:

- How many times EverAnalyzer was able to give the faster framework for each of the 30 given datasets of the experiment.
- How many times was EverAnalyzer was able to give continuous right answers in the span of the 30 given datasets of the experiment.
- How many correct answers EverAnalyzer gave in each consecutive sequence of correct answers.

The execution speed of each data processing is described in Table 11.

Table 11 – Processing Execution Speeds

Disease/Condition	Spark Execution Speed (milliseconds)	Hadoop Execution Speed (milliseconds)
Anaemia	11185	1387
Cancer	649	1369
Cholera	469	1456
Coronavirus	474	1439
Influenza	552	1414
Monkeypox	531	1358
Obesity	489	1320
Pneumonia	424	1327
Smallpox	431	1416
Syphilis	427	1438
Tetanus	388	1388
Yellow fever	392	1401
Zika virus	389	1300
Trachoma	354	1472
Diabetes	480	1426
Diarrhoea	374	1343
Ebola virus	496	1434
Epilepsy	434	1420
Hepatitis	325	1418
HIV-AIDS	485	1426
Depression	345	1510
Disability	279	1482
Cardiovascular	291	1346
Chagas	351	1469
Dementia	365	1330
Dracunculiasis	292	1430
Echinococcosis	376	1344
Foodborne	384	1441
Hypertension	382	1522
Infertility	352	1317

Analyzing the information of Tables 10 and 11, the following results are extracted. Firstly, the case of the literature where Spark has better performances for small dataset sizes is proved. Spark was not fast enough to exceed Hadoop in terms of execution Speed only in the first experiment (Anaemia) mostly because of the time that Spark needed to start up. Secondly, EverAnalyzer responded with the most efficient proposed framework 24/30 times (80% success). Of course, the expectation is that putting even more datasets will increase this percentage. More specifically, Figure 43 shows the consecutive correct sequences, until a bad proposal is found. In the graph

the x-axis shows the number of each consecutive sequence of correct answers, and the y-axis shows the amount of consecutive correct suggestions. Although the last pillar has 7 consecutive correct answers, it could probably be even bigger if more data had been collected, as no failed suggestion was found in the last streak, but the data just ran out. As the graph shows, the continued success of EverAnalyzer seems to be followed by increasing monotony, which means that for more data it will be a percentage of far more than 80%.

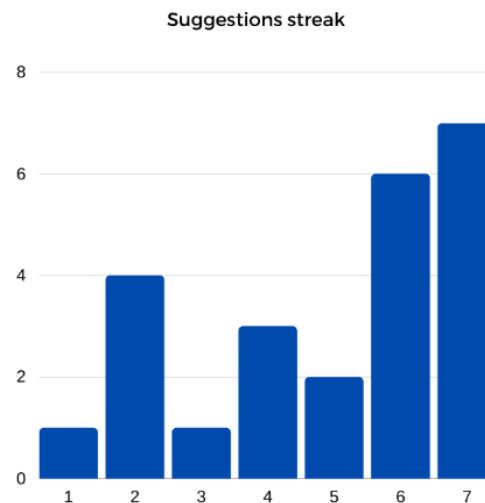


Figure 43 – Correct suggestions streak

In an ideal system, the end-user will always use the most efficient framework for his analysis, so the ideal time for his analysis would be exactly the time for the optimal framework of the two, to execute the analytic processes on his own use case. An analyst who does not know the best framework for his analysis, could try to maximize his chances of choosing the best framework between the two, each time tossing a coin to choose which one to use.

Figure 44 shows the best execute time (red) of the two frameworks for each dataset analysis done, along with the worst (blue) and their difference, in milliseconds. Therefore, the red line shows the ideal time that a User could do all 30 analyzes, while the orange line shows the time that gets “saved” from the worst case, doing the subtraction worst – best. It is easy to see in the graph that the orange line, if we draw vertical lines from the x-axis to the ends of the field that defines it, has an area over half of the corresponding area of the blue line. This means from a mathematical point of view, that the user in such a case saves from the bad cases, more than half of his total time if he constantly takes the best framework as a decision.

Figure 45, on the other hand, shows again the ideal (red) and worst (blue) analysis times, but this time, it also shows the time an analyst would do if he listened consistently to the EverAnalyzer (orange) proposals through the experimentation that was held. This Figure combines the information presented in Figures 43 and 44, by showing exactly the times that EverAnalyzer failed to make a good suggestion. Also, it is visible that EverAnalyzer follows 80% of the time the best execution time.

Finally, due to the good performance of EverAnalyzer (80%) in the proposal of the fastest framework for each given dataset, it can also be seen in the last image that EverAnalyzer really manages to reduce the waiting time of its users for the processing jobs carried out within it. Because EverAnalyzer is data-driven, the more metadata it

manages to collect, the closer it will get to the ideal red line in its execution speeds. Of course, it is also obvious that the system used by EverAnalyzer has the negative that if it happens and makes the bad suggestion, then the user is forced to wait for the worst time to perform his analysis.

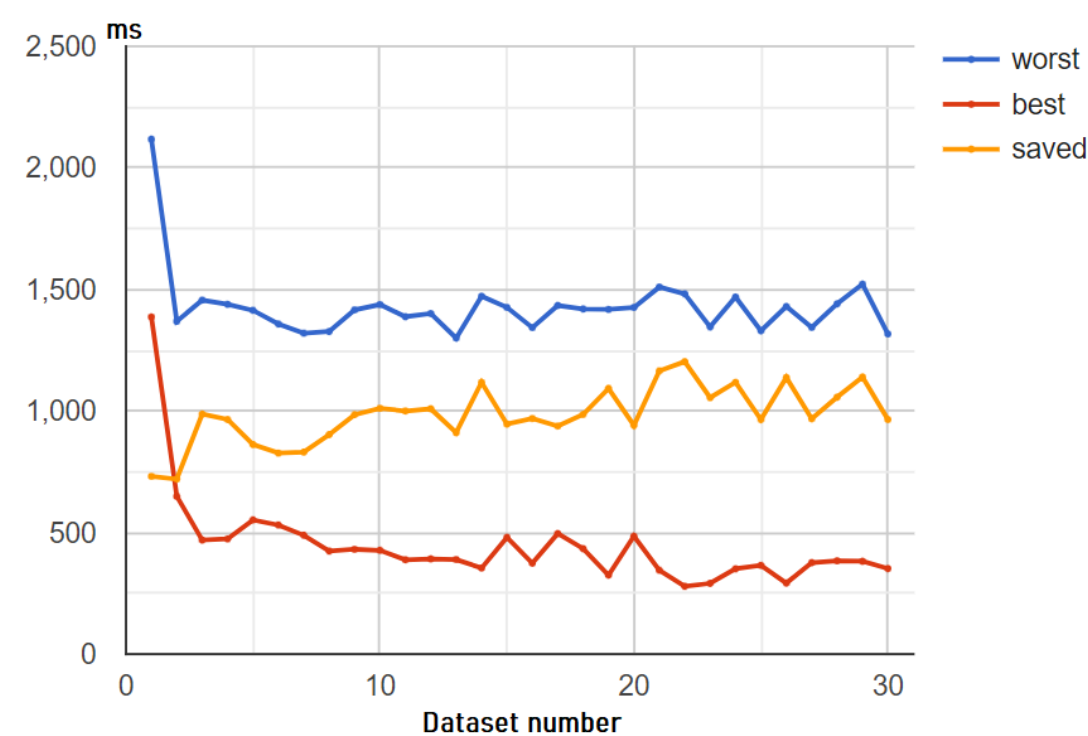


Figure 44 – Worst-Best time difference

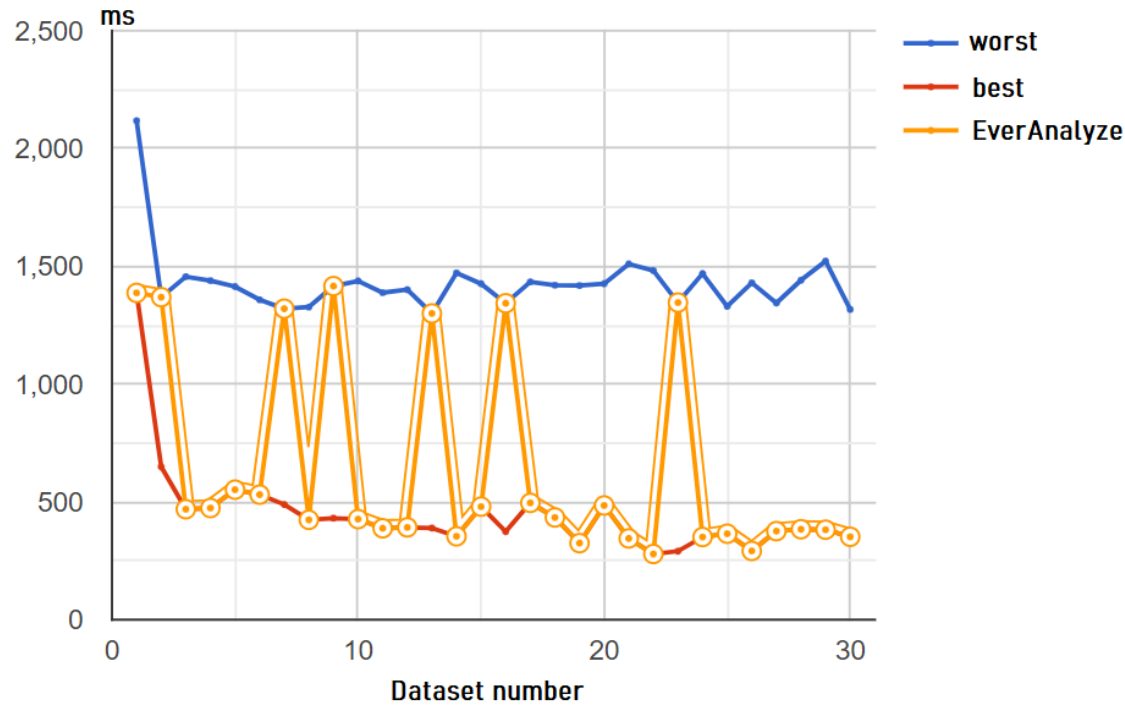


Figure 45 – Worst-Best EverAnalyzer

6. Conclusions and Next Steps

In this Thesis, a review was conducted in the Big Data industry, in the tools for their management and analysis, as well as in the ways of their analysis. A literature review was also performed on two of the most popular Big Data management and analysis tools, namely Hadoop and Spark, as well as their respective libraries for ML namely Mahout and Spark MLlib. In addition, a Web Application for Big Data Management called EverAnalyzer was built. This application is a platform that aims to manage end-user analyzes as fully error-tolerant, not letting them make mistakes, using user-friendly warnings. In addition to its fault tolerant logic, it gives its users the ability to collect, pre-process, process, analyze and access all their data. The way EverAnalyzer is designed makes it a better experience for any analyst who wants to do text mining on text data, via Twitter, which is one of the most popular social media right now. Finally, experiments were performed on the mentioned platform, as well as interpretation of the results, with the aim of examining the success rates of the framework suggestion functionality for Wordcount jobs to its end-users.

the literature, Spark appeared to outperform Hadoop in smaller data, as well as in data that is smaller than the total RAM of the system that manages them using Spark MLlib. But Hadoop seems to outperform larger data, just as Mahout outperforms Spark MLlib in data larger than the total system RAM used. Finally, Spark seems to be less fault tolerant than Hadoop, resulting in a friendlier use of Hadoop by those interested in these frameworks even though it is not always the best choice in terms of analysis speed.

The EverAnalyzer platform uses the literature developed to give a better experience to its end-users. Not many platforms that use a combination of such frameworks do seem to exist. But such systems are what they are supposed to innovate in the field of Big Data analysis, giving optimal solutions to data analysts. The mentioned systems should keep in mind that a person has limited time. It is therefore difficult for an end-user to learn every area of Data Science, from building Big Data software to interpreting and conducting Big Data experiments. EverAnalyzer is one such system, which helps with the above problem, solving the hands of Data Analysts, Data Scientists and Data Engineers. Obviously, the mentioned platform is a small step regarding the huge gap for innovation within the Big Data Management industry. It would be interesting to build platforms like EverAnalyzer, trying to add software and functionalities. Through them, the end-users of such systems will be given the ability to make their jobs easier. Such additions could be the development of software for more pre-processing jobs such as the ability to rename fields within JSON and more. But also, simpler additions such as the ability to collect data through other social media such as Facebook or Instagram.

Finally, the experimentation carried out using health data and the functionality of the framework proposal for processing jobs of EverAnalyzer, seems to have been promising for very positive results. It would be very interesting for this experiment to be performed with an even larger number of datasets, but also more data collected per dataset. With such experimentation, it would be easier to see the exact success rate of EverAnalyzer proposals, using statistical confidence intervals. In addition, it would be interesting to experiment with a larger system, using more resources such as RAM and HDD. It would also be important to perform experiments within a cloud with more than one cluster nodes.

References

- [1] Total data volume worldwide 2010-2025. (n.d.). Statista. Retrieved June 13, 2022, from <https://www.statista.com/statistics/871513/worldwide-data-created/>
- [2] Kulkarni, R. (2019, February 7). Big data goes big. Forbes. <https://www.forbes.com/sites/rkulkarni/2019/02/07/big-data-goes-big/?sh=5b985d0920d7>
- [3] Bhosale, H. S., & Gadekar, D. P. (2014). A review paper on big data and hadoop. *International Journal of Scientific and Research Publications*, 4(10), 1-7.
- [4] Apache Hadoop. (n.d.). Apache.Org. Retrieved June 13, 2022, from <https://hadoop.apache.org/>
- [5] Apache Spark™ - Unified Engine for large-scale data analytics. (n.d.). Apache.Org. Retrieved June 13, 2022, from <https://spark.apache.org/>
- [6] Ahmed, N., Barczak, A. L., Susnjak, T., & Rashid, M. A. (2020). A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale data sets using HiBench. *Journal of Big Data*, 7(1), 1-18.
- [7] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., Mccauley, M., ... & Stoica, I. (2012). Fast and interactive analytics over Hadoop data with Spark. *Usenix Login*, 37(4), 45-51.
- [8] Ahmadvand, H., Goudarzi, M., & Foroutan, F. (2019). Gapprox: using gallup approach for approximation in big data processing. *Journal of Big Data*, 6(1), 1-24.
- [9] Samadi, Y., Zbakh, M., & Tadonki, C. (2018). Performance comparison between Hadoop and Spark frameworks using HiBench benchmarks. *Concurrency and Computation: Practice and Experience*, 30(12), e4367.
- [10] Musselman, A. (2018). Apache Mahout. In *Encyclopedia of Big Data Technologies* (pp. 1–5). Springer International Publishing.
- [11] MLlib. (n.d.). Apache.Org. Retrieved June 13, 2022, from <https://spark.apache.org/mllib/>
- [12] Aziz, K., Zaidouni, D., & Bellafkih, M. (2018, October). Big data processing using machine learning algorithms: Mllib and mahout use case. In *Proceedings of the 12th international conference on intelligent systems: theories and applications* (pp. 1-6).
- [13] Jin, X., Wah, B. W., Cheng, X., & Wang, Y. (2015). Significance and challenges of big data research. *Big data research*, 2(2), 59-64.
- [14] Murthy, A. (2014). *Apache Hadoop yarn: Moving beyond MapReduce and batch processing with Apache Hadoop 2*. Addison-Wesley Professional. <https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>
- [15] Garg, N. (2013). *Apache Kafka*. Packt Publishing. <https://kafka.apache.org/>
- [16] Welcome to Apache flume — Apache flume. (n.d.). Apache.org. Retrieved July 22, 2022, from <https://flume.apache.org/>
- [17] MongoDB: The developer data platform. (n.d.). MongoDB. Retrieved July 22, 2022, from <https://www.mongodb.com/>
- [18] Documentation home. (n.d.). Twitter. Retrieved June 13, 2022, from <https://developer.twitter.com/en/docs>
- [19] Bagga, S., & Sharma, A. (2018, August). Big data and its challenges: a review. In *2018 4th International Conference on Computing Sciences (ICCS)* (pp. 183-187). IEEE.
- [20] Koo, J., Kang, G., & Kim, Y. G. (2020). Security and privacy in big data life cycle: a survey and open challenges. *Sustainability*, 12(24), 10571.

- [21] Liu, J., Li, J., Li, W., & Wu, J. (2016). Rethinking big data: A review on the data quality and usage issues. *ISPRS journal of photogrammetry and remote sensing*, 115, 134-142.
- [22] Goudarzi, M. (2017). Heterogeneous architectures for big data batch processing in mapreduce paradigm. *IEEE Transactions on Big Data*, 5(1), 18-33.
- [23] Anderson, J. W., Kennedy, K. E., Ngo, L. B., Luckow, A., & Apon, A. W. (2014, October). Synthetic data generation for the internet of things. In *2014 IEEE International Conference on Big Data (Big Data)* (pp. 171-176). IEEE.
- [24] Watson, D., & Riden, J. (2008, April). The honeynet project: Data collection tools, infrastructure, archives and analysis. In *2008 WOMBAT Workshop on Information Security Threats Data Collection and Sharing* (pp. 24-30). IEEE.
- [25] Shapira, Gwen, and Jeff Holoman. "Flafka: Apache flume meets apache kafka for event processing." (2014).
- [26] Padgavankar, M. H., & Gupta, S. R. (2014). Big data storage and challenges. *International Journal of Computer Science and Information Technologies*, 5(2), 2218-2223.
- [27] Strauch, C., Sites, U. L. S., & Kriha, W. (2011). NoSQL databases. *Lecture Notes, Stuttgart Media University*, 20(24), 79.
- [28] Mehdipour, F., Noori, H., & Javadi, B. (2016). Energy-efficient big data analytics in datacenters. In *Advances in Computers* (Vol. 100, pp. 59-101). Elsevier.
- [29] Garg, S. (2018). Dynamic Power Management for Dark Silicon Multicore Processors. *Advances in Computers*, 110, 171-216.
- [30] Khezr, S. N., & Navimipour, N. J. (2017). MapReduce and its applications, challenges, and architecture: a comprehensive review and directions for future research. *Journal of Grid Computing*, 15(3), 295-321.
- [31] Olston, C., Reed, B., Srivastava, U., Kumar, R., & Tomkins, A. (2008, June). Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 1099-1110).
- [32] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., ... & Murthy, R. (2009). Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2), 1626-1629.
- [33] Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F. B., & Babu, S. (2011, January). Starfish: A Self-tuning System for Big Data Analytics. In *Cidr* (Vol. 11, No. 2011, pp. 261-272).
- [34] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*.
- [35] Isard, M., Budiu, M., Yu, Y., Birrell, A., & Fetterly, D. (2007, March). Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007* (pp. 59-72).
- [36] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., & Czajkowski, G. (2010, June). Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (pp. 135-146).
- [37] Chintapalli, S., Dagit, D., Evans, B., Farivar, R., Graves, T., Holderbaugh, M., ... & Poulosky, P. (2016, May). Benchmarking streaming computation engines: Storm, flink and spark streaming. In *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)* (pp. 1789-1792). IEEE.
- [38] Jason, M. (2004). Apache Storm. *Signet Book*.

- [39] Apache flink: Stateful computations over data streams. (n.d.). Apache.org. Retrieved July 22, 2022, from <http://flink.apache.org/>
- [40] Apache Spark™ - Unified Engine for large-scale data analytics. (n.d.). Apache.org. Retrieved July 22, 2022, from <http://spark.apache.org/>
- [41] Noghabi, S. A., Paramasivam, K., Pan, Y., Ramesh, N., Bringham, J., Gupta, I., & Campbell, R. H. (2017). Samza: Stateful scalable stream processing at LinkedIn. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, 10(12), 1634–1645. <https://doi.org/10.14778/3137765.3137770>
- [42] Apache Software Foundation. (n.d.). Apache apex. Apache.org. Retrieved July 22, 2022, from <http://apex.incubator.apache.org/>
- [43] Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R. J., Lax, R., ... & Whittle, S. (2015). The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing.
- [44] Ji, C., AWADA, U., & LI, K. (2013). Big data processing: Big challenges. *Journal of Interconnection Network*, 13.
- [45] Aziz, K., Zaidouni, D., & Bellafkih, M. (2018, April). Real-time data analysis using Spark and Hadoop. In *2018 4th international conference on optimization and applications (ICOA)* (pp. 1-6). IEEE.
- [46] Pirzadeh, P. (2015). On the performance evaluation of big data systems (Doctoral dissertation, UC Irvine).
- [47] Mavridis, I., & Karatza, E. (2015). Log file analysis in cloud with Apache Hadoop and Apache Spark.
- [48] Kuo, A., Chrimes, D., Qin, P., & Zamani, H. (2019, January). A Hadoop/MapReduce Based Platform for Supporting Health Big Data Analytics. In *ITCH* (pp. 229-235).
- [49] Hazarika, A. V., Ram, G. J. S. R., & Jain, E. (2017, February). Performance comparison of Hadoop and spark engine. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)* (pp. 671-674). IEEE.
- [50] Ji, C., Li, Y., Qiu, W., Awada, U., & Li, K. (2012, December). Big data processing in cloud computing environments. In *2012 12th international symposium on pervasive systems, algorithms and networks* (pp. 17-23). IEEE.
- [51] Habeeb, R. A. A., Nasaruddin, F., Gani, A., Hashem, I. A. T., Ahmed, E., & Imran, M. (2019). Real-time big data processing for anomaly detection: A survey. *International Journal of Information Management*, 45, 289-307.
- [52] Liu, L. (2013). Computing infrastructure for big data processing. *Frontiers of Computer Science*, 7(2), 165-170.
- [53] Rajabion, L., Shaltooli, A. A., Taghikhah, M., Ghasemi, A., & Badfar, A. (2019). Healthcare big data processing mechanisms: The role of cloud computing. *International Journal of Information Management*, 49, 271-289.
- [54] Yao, Q., Tian, Y., Li, P. F., Tian, L. L., Qian, Y. M., & Li, J. S. (2015). Design and development of a medical big data processing system based on Hadoop. *Journal of medical systems*, 39(3), 1-11.
- [55] MapReduce Tutorial. (n.d.). Apache.Org. Retrieved June 13, 2022, from <https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html>
- [56] Akil, B., Zhou, Y., & Röhm, U. (2017, December). On the usability of Hadoop MapReduce, Apache Spark & Apache flink for data science. In *2017 IEEE International Conference on Big Data (Big Data)* (pp. 303-310). IEEE.

- [57] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., ... & Stoica, I. (2012). Resilient Distributed Datasets: A {Fault-Tolerant} Abstraction for {In-Memory} Cluster Computing. In 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12) (pp. 15-28).
- [58] Soltanpoor, R., & Sellis, T. (2016, September). Prescriptive analytics for big data. In Australasian database conference (pp. 245-256). Springer, Cham
- [59] Petre, R. S. (2012). Data mining in cloud computing. Database Systems Journal, 3(3), 67-71.
- [60] SangeethaLakshmi, M. G., & Jayashree, M. M. (2019). Comparative Analysis of Various Tools for Data Mining and Big Data Mining. International Research Journal of Engineering and Technology (IRJET), 6(4), 704-708.
- [61] Bioinformatics Laboratory, & University of Ljubljana. (n.d.). Data Mining. Orangedatamining.com. Retrieved July 22, 2022, from <https://orangedatamining.com/>
- [62] Ripley, B. D. (2001). The R project in statistical computing. MSOR Connections, 1(1), 23–25. <https://doi.org/10.11120/msor.2001.01010023>
- [63] Weka a3 - data mining with open source machine learning software in java. (n.d.). Waikato.Ac.Nz. Retrieved July 22, 2022, from <https://www.cs.waikato.ac.nz/ml/weka/>
- [64] shogun: Shōgun. (n.d.).
- [65] Lee, P. (2022, March 5). RapidMiner. RapidMiner. <https://rapidminer.com/>
- [66] Explainable AI platform. (n.d.). Neuraldesigner.com. Retrieved July 22, 2022, from <https://www.neuraldesigner.com/>
- [67] MikePlumleyMSFT. (n.d.). Introduction to SharePoint - SharePoint in Microsoft 365. Microsoft.com. Retrieved July 22, 2022, from <https://docs.microsoft.com/en-us/sharepoint/introduction>
- [68] IBM Cognos Analytics. (n.d.). IBM Cognos Analytics. Retrieved July 22, 2022, from <https://www.ibm.com/products/cognos-analytics>
- [69] Board: Intelligent planning for finance, supply chain, & retail. (n.d.). Board. Retrieved July 22, 2022, from <https://www.board.com/en>
- [70] Sisense - infuse analytics everywhere. (2021, October 21). Sisense. <https://www.sisense.com/>
- [71] KEEL: A software tool to assess evolutionary algorithms for Data Mining problems (regression, classification, clustering, pattern mining and so on). (n.d.). Ugr.es. Retrieved July 22, 2022, from <https://sci2s.ugr.es/keel/index.php>
- [72] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... & Talwalkar, A. (2016). Mllib: Machine learning in apache spark. The Journal of Machine Learning Research, 17(1), 1235-1241.
- [73] Curtis, A. R., Kim, W., & Yalagandula, P. (2011, April). Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In 2011 Proceedings IEEE INFOCOM (pp. 1629-1637). IEEE.
- [74] Anil, R., Çapan, G., Drost-Fromm, I., Dunning, T., Friedman, E., Grant, T., ... & Yilmazel, Ö. (2020). Apache Mahout: Machine Learning on Distributed Dataflow Systems. J. Mach. Learn. Res., 21(127), 1-6.
- [75] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... & Talwalkar, A. (2016). Mllib: Machine learning in apache spark. The Journal of Machine Learning Research, 17(1), 1235-1241.
- [76] Mostafaeipour, A., Jahangard Rafsanjani, A., Ahmadi, M., & Arockia Dhanraj, J. (2021). Investigating the performance of Hadoop and Spark platforms on machine learning algorithms. The Journal of Supercomputing, 77(2), 1273-1300.

- [77] Richter, A. N., Khoshgoftaar, T. M., Landset, S., & Hasanin, T. (2015, August). A multi-dimensional comparison of toolkits for machine learning with big data. In 2015 IEEE International Conference on Information Reuse and Integration (pp. 1-8). IEEE.
- [78] Huang, X., Jiang, P., & Ma, J. (2018, July). A machine learning application for electric power industrial big-data based on Hadoop. In 2018 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD) (pp. 1134-1137). IEEE.
- [79] Wan, J., Tang, S., Li, D., Wang, S., Liu, C., Abbas, H., & Vasilakos, A. V. (2017). A manufacturing big data solution for active preventive maintenance. *IEEE Transactions on Industrial Informatics*, 13(4), 2039-2047.
- [80] Maktoubian, J. (2017). Proposing a streaming Big Data analytics (SBDA) platform for condition based maintenance (CBM) and monitoring transportation systems. *EAI Endorsed Transactions on Scalable Information Systems*, 4(13).
- [81] Gupta, N., Lenka, R. K., Barik, R. K., & Dubey, H. (2017). Fair: A hadoop-based hybrid model for faculty information retrieval system. *arXiv preprint arXiv:1706.08018*.
- [82] Faghri, F., Hashemi, S. H., Babaeizadeh, M., Nalls, M. A., Sinha, S., & Campbell, R. H. (2017). Toward scalable machine learning and data mining: the bioinformatics case. *arXiv preprint arXiv:1710.00112*.
- [83] Health topics. (n.d.). Who.Int. Retrieved June 13, 2022, from <https://www.who.int/europe/health-topics?>