

ŽILINSKÁ UNIVERZITA  
V ŽILINE

Faculty of Electrical Engineering and Information Technology

Distributed Big Data Processing Capabilities in Continental's  
Environment

Diploma Thesis

**Matej Vrábek, B.S.**

Study programme: Process Management

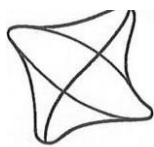
Field of study: Cybernetics

Training department: University of Žilina, Supervisor:

doc. Ing. Peter Peniak, PhD.

Consultant: Solano Mora, Erick, PhD.

Žilina 2024



UNIVERSITY OF ŽILINA IN ŽILINA  
Faculty of Electrical Engineering  
and Information Technologies  
Department of Management  
and Information Systems

Academic year 2023/2024

Registration CFSLO 28260220242032

## ASSIGNMENT OF THE DIPLOMA THESIS

Student Name: **Matej Vrabel**  
Study program: **Process control**  
Master's thesis: **Possibilities of Distributed Big Data Processing at Continental**

### Instructions for the preparation of the diploma thesis:

1. Distributed processing of big data through selected programming languages (Python, Scala,...).
2. Available environments for distributed data processing (Apache Spark, Ray framework).
3. Experimental verification of the possibilities of distributed processing on the available data of the selected Continental plant (sensor data) and comparison of the processing results.
4. Implementation of the selected solution under Continental conditions.

Supervisor of the diploma thesis : **doc. Ing. Peter Peniak, PhD.**

Thesis submission date : **30.04.2024**

In Žilina on 30.10.2023

Prof. And j **IF**  
**D.**  
**t'** Department

## **Affidavit**

I declare that I have prepared the assigned diploma thesis independently, under the professional guidance of the supervisor of the thesis, who was Associate Professor Ing. Peter Peniak, PhD., and I have used only the literature stated in the thesis.

Žilina, April 30, 2024

signature

## Thanks

I would like to thank my supervisor, Associate Professor Ing. Peter Peniak, PhD., for his willingness, factual advice when writing the diploma thesis and for his friendly approach.

My huge thanks go to the consultant, Erick Solano Mora, PhD., a data scientist, thanks to whom I managed to implement the practical part. I thank him for his direction, willingness, communication, and for the experience he passed on to me.

I would also like to thank the teachers from the entire Department of Control and Information Systems, for passing on a lot of information to our lives, for their human approach, thanks to which I felt comfortable at the university.

My thanks go to my mother, who stood by me during the creation of my diploma thesis. I also thank all my family and friends who encouraged me and gave me the strength I needed to finish.

## Abstract

The diploma thesis deals with the field of big data processing, which requires a specific approach for the given purpose. It theoretically describes the tools that are used in this area and practically shows their application for processing process data of the Continental manufacturing company. The aim of the thesis was to conduct a research of the *Python* and *Scala* programming languages, as well as *the Apache Spark* and *Ray* environments for distributed processing of big data. Another goal was to create an experimental workplace in the Continental data lake and use it in the processing of the company's production data. To implement the later task, work with *a cluster* into which the *Apache Spark environment was installed* was incorporated. Programs written by a consultant in the *Scala* programming language were also used. The output of the work is the results of the analysis summarized in tables and graphs in the fifth chapter of the documentation.

**Keywords:** big data, Python, Scala, Apache Spark, cluster

## Abstract

Diploma thesis deals with a field of Big Data processing, which require a specific approach for this purpose. It theoretically describes tools, which are used in mentioned field and practically shows their application for processing of process data of the production plant of the Continental company. Aim of the work was to do a research of programming languages *Python* and *Scala*, as well of frameworks *Apache Spark* and *Ray* for distributed Big Data processing. Another aim was the creation of experimental workplace in a data lake of the Continental company and its use in the processing of production data of the mentioned company. For a realization of the latter mentioned task was incorporated a work with *cluster*, into which was installed a framework *Apache Spark*. There were also used programs written in *Scala* programming language by a consultant of thesis. Output of thesis are results from analysis summarized in tables and graphs in the fifth chapter of documentation.

**Keywords:** Big Data, Python, Scala, Apache Spark, cluster

## Contents

INTRODUCTION .....	1
1 PROGRAMMING LANGUAGES ON DISTRIBUTED DATA PROCESSING.....	3
1.1 Python 3	
1.1.1 Python and big data .....	4
1.1.2 Python libraries used in the field of data analysis .....	6
1.2 Scala 18	
1.3 Python vs. Scala.....	19
1.3.1 Performance.....	19
1.3.2 Error.....	19
1.3.3 Project size .....	19
1.3.4 Hadoop.....	19
1.3.5 Apache Spark.....	20
2 AVAILABLE ENVIRONMENT FOR DISTRIBUTED DATA PROCESSING.....	21
2.1 Apache Spark.....	21
2.1.1 Properties.....	22
2.1.2 Composition .....	23
2.1.3 Principle of operation.....	24
2.1.4 Work modes.....	27
2.1.5 Exploitation.....	27
2.2 Ray 28	
2.2.1 Properties.....	28
2.2.2 Libraries .....	29
3 SERVICES FOR WORKING WITH BIG DATA.....	30
3.1 Data Lakes .....	30
3.1.1 Variety of data stored .....	30
3.1.2 Data Lake Location .....	31
3.2 AWS – Amazon Web Services .....	32
3.2.1 S3 – Simple Storage Service.....	32
3.2.2 EC2 – Elastic Compute Cloud .....	34
3.2.3 EMR – Elastic MapReduce.....	38
4 EXPERIMENTAL WORKPLACE.....	40
4.1 Design of an experimental workplace .....	40
4.2 Creation of an experimental workplace.....	41

4.2.1	Instance.....	42
4.2.2	Dataset.....	47
4.2.3	Cluster.....	49
4.3	Data processing of the manufacturing company .....	51
5	RESULTS OF DATA ANALYSIS .....	59
5.1	Vulcanization or Curing .....	59
5.2	Extrusion.....	70
5.3	Tire Molding.....	80
	CONCLUSION.....	87



## List of images

Fig. 1.1 Comparison of the work of the compiler and the interpreter [4] .....	4
Fig. 1.2 Pivoting process [11] .....	7
Fig. 1.3 <i>Broadcasting Two NumPy</i> fields of various shapes [13].....	8
Fig. 1.4 Comparison of properties of different classifiers, part a [16].....	9
Fig. 1.5 Comparison of properties of different classifiers, part b [16].....	9
Fig. 1.6 Regression result using the decision tree method and <i>AdaBoost</i> [18] .....	10
Fig. 1.7 Comparison of properties of different clustering algorithms, part a [19].....	11
Fig. 1.8 Comparison of properties of different clustering algorithms, part b [19].....	11
Fig. 1.9 Comparison of model accuracy on training and test data [21].....	12
Fig. 1.10 Drawing a line using the function <code>plot()</code> [25].....	14
Fig. 1.11 Plotting a Bar Chart Using the Function <code>bar()</code> [25].....	15
Fig. 1.12 Graph showing a network using nodes and edges between them [29].....	16
Fig. 1.13 Function usage <code>Symbol()</code> to create symbolic variables [30].....	16
Fig. 1.14 Output from the example shown in Figure 1.13 [30] .....	17
Fig. 1.15 Reading the selected dataset using the <code>read_csv()</code> function of the library <i>pandas</i> [31].....	17
Fig. 1.16 Reading the same dataset using the <code>read_csv()</code> function of the library <i>dask</i> [31].....	17
Fig. 2.1 Composition <i>Apache Spark</i> [50].....	24
Fig. 2.2 Division of RDD into partitions [51] .....	25
Fig. 2.3 RDD operations [51] .....	25
Fig. 2.4 Architecture <i>Apache Spark</i> [51] .....	26
Fig. 3.1 List of bucket objects <i>mybucket</i> [64] .....	34
Fig. 3.2 Starting instances from AMI [68].....	35
Fig. 3.3 Nomenclature of the EC2 instance [69].....	37
Figure 3.4 Life cycle of an EC2 instance [70] .....	37
Figure 3.5 Elasticity of computational capacities [73].....	39
Fig. 4.1 Design of the experimental workplace.....	40
Fig. 4.2 Login window to the Continental data lake .....	41
Figure 4.3 Continental's Data Lake .....	41
Fig. 4.4 Preview of instances in the Continental data lake environment.....	43
Fig. 4.5 Selection of the area of use of an instance in its creation .....	43
Fig. 4.6 Instance configuration .....	44
Fig. 4.7 Type of EC2 instance created by us .....	45

Fig. 4.8 Instance states .....	45
Fig. 4.9 Graphical environment <i>DeepLearning</i> .....	46
Fig. 4.10 SSH console.....	46
Fig. 4.11 S3 buckets in the Continental Data Lake.....	47
Fig. 4.12 Approximation of the dataset from the production plant in Korbach.....	48
Fig. 4.13 Command to list all items of the selected S3 bucket.....	48
Fig. 4.14 Listing of individual objects S3 buckets of the production plant in Korbach together with summarization.....	49
Fig. 4.15 Preview of <i>Clustre</i> in Continental's data lake .....	50
Fig. 4.16 Formation of EMR <i>Clusters</i> .....	50
Fig. 4.17 List of created <i>clusters</i> .....	51
Fig. 4.18 Information on <i>Clusters emr_vrabel</i> .....	52
Fig. 4.19 Connecting to <i>Cluster</i> via SSH protocol in MobaXterm software.....	52
Fig. 4.20 Start-up <i>Apache Spark</i> in EMR <i>Clusters</i> .....	53
Fig. 4.21 Part <i>GitHub</i> Code <i>write_data_summaries.scala</i> .....	54
Fig. 4.22 Importing libraries and creating a connection.....	54
Fig. 4.23 List of parameters from the production branch of the press shop.....	55
Fig. 4.24 Loading all data from the stamping industry.....	55
Fig. 4.25 List of stations with the number of values in the production area of the press shop .....	55
Fig. 4.26 Total number of values for individual parameters of the CU306 station .....	56
Fig. 4.27 Modification of the selected two parameters of the CU306 station together with the command to extract the summarization .....	56
Fig. 4.28 Extract of summarization of selected two parameters of the CU306 station..	57
Fig. 4.29 List of stations with the number of values in the extrusion production area...	58
Fig. 4.30 List of stations with the number of values in the ready-to-wear production area .....	58
Fig. 4.31 Parameters of the TBPH2 station with values <i>Null</i> .....	58
Fig. 5.1 Box diagram of stations CU306, CU307, CU310 for parameter b) .....	61
Fig. 5.2 Box diagram of stations CU306, CU307, CU310 for parameter c) .....	62
Fig. 5.3 Box diagram of stations CU306, CU307, CU310 for parameter d) .....	63
Fig. 5.4 Box diagram of stations CU306, CU307, CU310 for parameter e) .....	64
Fig. 5.5 Box diagram of stations CU306, CU307, CU310 for parameter f) .....	65
Fig. 5.6 Box diagram of stations CU306, CU307, CU310 for parameter g) .....	66
Fig. 5.7 Box diagram of stations CU306, CU307, CU310 for parameter i) .....	68
Fig. 5.8 Box diagram of stations CU306, CU307, CU310 for parameter j) .....	69
Fig. 5.9 Box diagram of the EX071 station for parameter a) .....	70
Fig. 5.10 Box diagram of the EX071 station for parameter b).....	71

---

Fig. 5.11 Box diagram of the EX071 station for parameter c) .....	72
Fig. 5.12 Box diagram of the EX071 station for parameter d).....	73
Fig. 5.13 Box diagram of the EX071 station for parameter e).....	74
Fig. 5.14 Box diagram of the EX071 station for parameter g).....	76
Fig. 5.15 Box diagram of the EX071 station for parameter h).....	77
Fig. 5.16 Box diagram of the EX071 station for parameter i) .....	78
Fig. 5.17 Box diagram of the EX071 station for parameter j) .....	79
Fig. 5.18 Box diagram of stations TBPH1, TBPH2 for parameter a).....	80
Fig. 5.19 Box diagram of stations TBPH1, TBPH2 for parameter b).....	81
Fig. 5.20 Box diagram of stations TBPH1, TBPH2 for parameter c).....	82
Fig. 5.21 Box diagram of TBPH1 stations for parameter d).....	83
Fig. 5.22 Box diagram of stations TBPH1, TBPH2 for parameter e).....	84
Fig. 5.23 Box diagram of stations TBPH1, TBPH2 for parameter f).....	85

## List of tables

Table 3.1 Meaning of EC2 instance nomenclature [69] .....	36
Table 4.1 Names of programs used by the consultant.....	53
Table 5.1 Analysis of stations CU306, CU307, CU310 for parameter a) .....	59
Table 5.2 Analysis of stations CU306, CU307, CU310 for parameter b) .....	60
Table 5.3 Analysis of stations CU306, CU307, CU310 for parameter c) .....	62
Table 5.4 Analysis of stations CU306, CU307, CU310 for parameter d) .....	63
Table 5.5 Analysis of stations CU306, CU307, CU310 for parameter e) .....	64
Table 5.6 Analysis of stations CU306, CU307, CU310 for parameter f) .....	65
Table 5.7 Analysis of stations CU306, CU307, CU310 for parameter g) .....	66
Table 5.8 Analysis of stations CU306, CU307, CU310 for parameter h) .....	67
Table 5.9 Analysis of stations CU306, CU307, CU310 for parameter i).....	68
Table 5.10 Analysis of stations CU306, CU307, CU310 for parameter j) .....	69
Table 5.11 Analysis of the EX071 station for parameter a) .....	70
Table 5.12 Analysis of the EX071 station for parameter b) .....	71
Table 5.13 Analysis of the EX071 station for parameter c) .....	72
Table 5.14 Analysis of the EX071 station for parameter d) .....	73
Table 5.15 Analysis of the EX071 station for parameter e) .....	74
Table 5.16 Analysis of the EX071 station for parameter f) .....	75
Table 5.17 Analysis of the EX071 station for parameter g) .....	76
Table 5.18 Analysis of the EX071 station for parameter h) .....	77
Table 5.19 Analysis of the EX071 station for parameter i) .....	78
Table 5.20 Analysis of the EX071 station for parameter j) .....	79
Table 5.21 Analysis of TBPH1, TBPH2 stations for parameter a).....	80
Table 5.22 Analysis of TBPH1, TBPH2 stations for parameter b).....	81
Table 5.23 Analysis of TBPH1, TBPH2 stations for parameter c) .....	82
Table 5.24 Analysis of TBPH1, TBPH2 stations for parameter d).....	83
Table 5.25 Analysis of TBPH1, TBPH2 stations for parameter e).....	84
Table 5.26 Analysis of TBPH1, TBPH2 stations for parameter f).....	85

## List of abbreviations

Abbreviation	English Meaning	Slovak meaning
ACL	Access Control List	Access Control List
AI	Artificial Intelligence	Artificial intelligence
AMI	Amazon Machine Image	Amazon Outfit Image
API	Application Programming Interface	Application programming interface
AWS	Amazon Web Services	Amazon Web Services
CPU	Central Processing Unit	Central Processing Unit
DBMS	Database Management System	Database Management System
DBSCAN	Density-Based Spatial Clustering of Applications with Noise	Spatial clustering of dense-based noise-based applications
DLDA	Diagonal Linear Discriminant Analysis	Diagonal Linear Discriminant Analysis
DNS	Domain Name System	Domain Name System
DTW	Dynamic Time Warping	Dynamic Time Deformation
EBS	Elastic Block Store	Elastic Block Storage
EC2	Elastic Compute Cloud	Elastic cloud computing
EMR	Elastic MapReduce	Elastic MapReduce
ETL	Extract, Transform, Load	Extract, transform, load
FDA	Fisher's Discriminant Analysis	Analysis using Fisher's discriminant
FPGA	Field-programmable Gate Array	Programmable Gate Array
FTP	File Transfer Protocol	File transfer protocol
GCP	Google Cloud Platform	Google cloud platform
GNU	GNU's Not Unix	GNU is not Unix.
GPU	Graphics Processing Unit	Graphics processing unit
GUI	Graphical User Interface	Graphical user interface
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise	Spatial clustering of noise-based applications based on hierarchical density
HDFS	Hadoop Distributed File System	Hadoop Distributed File System
HTTP	Hypertext Transfer Protocol	Hypertext Transfer Protocol
ID	Identifier	Identifier
GOES	Integrated Development Environment	Integrated development environment
I/O	Input/Output	Input/Output
IoT	Internet of Things	Internet of Things
IP	Internet Protocol	Internet Protocol
JDBC	Java Database Connectivity	Java database connectivity

JSON	JavaScript Object Notation	Write JavaScript objects
LCS	Longest Common Subsequence	Longest common subsequence
LDA	Linear Discriminant Analysis	Linear Discrimination Analysis
LLVM	Low Level Virtual Machine	Low-level virtual machine
MIPy	Machine Learning Python	Python Machine Learning
NumPy	Numerical Python	Numerical Python
ODBC	Open Database Connectivity	Connectivity to an open database
OPTICS	Ordering points to identify the clustering structure	Arrangement of points to identify the clustering structure
ORC	Optimized Row Columnar	Optimized Column Row
AXIS	Operating System	Operating system
Pandas	Python Data Analysis	Python Data Analytics
PCA	Principal Component Analysis	Main Component Analysis
QDA	Quadratic Discriminant Analysis	Quadratic Discrimination Analysis
RAM	Random Access Memory	Random Access Memory
RBF SVM	Radial Basis Function Support Vector Machine	Support Vector Machine with radial base function
RDD	Resilient Distributed Dataset	Resilient distributed dataset
RDP	Remote Desktop Protocol	Remote Desktop Protocol
S3	Simple Storage Service	Simple storage service
Scala	Scalable language	Scalable language
SciPy	Scientific Python	Scientific Python
SDKs	Software Development Kits	Software Development Kits
SQL	Structured Query Language	Structuring Query Language
SRDA	Spectral Regression Discriminant Analysis	Spectral Regression Discriminant Analysis
SSH	Secure Shell	Safe packaging
stddev	Standard deviation	Deviation
SVM	Support Vector Machine	Support Vector Machine
SymPy	Symbolic Python	Symbolic Python
TPU	Tensor Processing Unit	Tensor processor
vCPU	virtual Centralized Processing Unit	Virtual Centralized Processing Unit
VNC	Virtual Network Computing	Virtual network computing

## INTRODUCTION

Data is obtained from every area around the world – whether it is manufacturing companies, financial or military sectors, all industries can move forward based on the collected data. Looking back at historical data, they can tell whether there has been a change for the better or for the worse compared to current data. In the case of manufacturing companies, it is also possible to evaluate the collected data in real time and decide whether the product should continue in the production process or, if it does not meet the required criteria, send it for reprocessing or waste. The above examples require data processing to obtain the required information. Based on their analyses, it is possible to improve the quality of the company or the areas to which the data relates. First of all, when analyzing data, you need to set a goal. In the case of a manufacturing company, it can be monitoring the condition of a specific production machine. Subsequently, it is necessary to have data collection secured. Once we have the data available, we can preprocess it – remove meaningless values from the data set, or fill in the missing values based on a certain heuristic. When we have the dataset ready, we can track and compare the values stored in it. In this way, we can use the comparison of historical data with current data to determine whether there is a deviation of the relevant parameters from the desired value. From this, it can then be concluded, for example, that maintenance is necessary for the machine. We can also look for dependencies between certain variables, or their values can be predicted using machine learning techniques. The possibilities of application are varied.

We are currently in the period of the fourth industrial revolution, which is characterized by a great boom in data. Machines in the production sphere are constantly producing new values, and with the growth of technological achievements, the speed of production of the aforementioned data is also increasing. We are talking here about the phenomenon of big data, which, in addition to its size and speed of production, can also be characterized by its heterogeneity. Big data requires a specific approach to both its storage and processing. Data lakes are used as storage, which do not adhere to the established form typical of relational databases. Environments using higher computing capacities in the form of combining multiple hardware components are used to process big data. Thanks to this, the desired processing results can be achieved in a reasonable time by implementing parallel calculations. In these environments, it is possible to interact with the data stored in the data lake using the functions of selected programming languages.

The thesis is divided into five chapters. In the first chapter, it deals with *the Python* and *Scala programming languages*, which can be applied in the field of big data processing. Here you can see the options that the languages offer, their advantages and disadvantages. The next chapter is devoted to *the Apache Spark and Ray environments* for distributed data processing. The reader has the opportunity to look into the principles on the basis of which we can quickly perform the desired tasks with data. Attention is mainly focused on *Apache Spark* because of its later use at work. The third chapter is devoted to data lakes and descriptions of AWS services that were applied in the thesis. These are data storage services and also the possibility of creating computing capacities that are assigned to the user of an account in *the cloud*. In the fourth chapter, the reader can see the creation of an experimental environment in the Continental data lake using the services described in the previous part of the thesis. The process that was used to process selected process data at the Continental production plant in Korbach is illustrated here. The last fifth chapter summarizes the results of the analysis of specific parameters from data sets belonging to three areas of production. It compares the results achieved between several stations of a specific production industry for the relevant parameters. The conclusion summarizes the individual areas of the diploma thesis, describes its main contribution to the data analysis of the aforementioned Continental manufacturing company and summarizes the phenomena that occurred during it.



# 1 PROGRAMMING LANGUAGES ON DISTRIBUTED DATA PROCESSING

To work with *big data*, the application of various methods is required. This includes data processing and distribution techniques. In practice, programming languages are used that already have the required functions implemented in their libraries. They allow us to load data of various formats and use modern approaches for their processing. Programmers do not have to create these functions from scratch, which is an advantage of the given programming languages. These include *Python* and *Scala*, which we will describe in the first chapter.

## 1.1 Python

*Python* is an interpreted, object-oriented, high-level programming language. It is very attractive for application development, as it contains high-level data structures and applies dynamic typing. Dynamic typing is based on the fact that the programmer does not have to assign the appropriate data type to variables when declaring them. The program is able to detect it during its runtime (unlike static programming languages, where checking the data type of variables occurs at compile time). *Python* also continues to be used as a scripting language or as a language for connecting existing components. It has a simple syntax that is easy to learn and emphasizes readability, reducing the cost of maintaining the program. *Python* supports modules and packages, increasing program modularity and code reusability [1], [2].

One of the reasons programmers like the aforementioned programming language is its increased productivity. *Python* does not include a compilation step, which makes the debugging cycle for editing and testing the program incredibly fast. Debugging programs is easy in *Python*. An error or bad input will never cause a segmentation error. This is an error that occurs when a program tries to access a memory location that it does not have permission to access (for example, indexing outside the boundaries of the created array). Instead, when the artist discovers an error, it throws an exception. An exception can be caught by the program if the programmer has adapted the source code written for it. Otherwise, the artist prints the tray track. The debugger is directly written in *Python* [1], [3].

Note: An interpreter is a program that directly executes the instructions of a high-level programming language without being translated into machine code (see figure 1.1) [4].

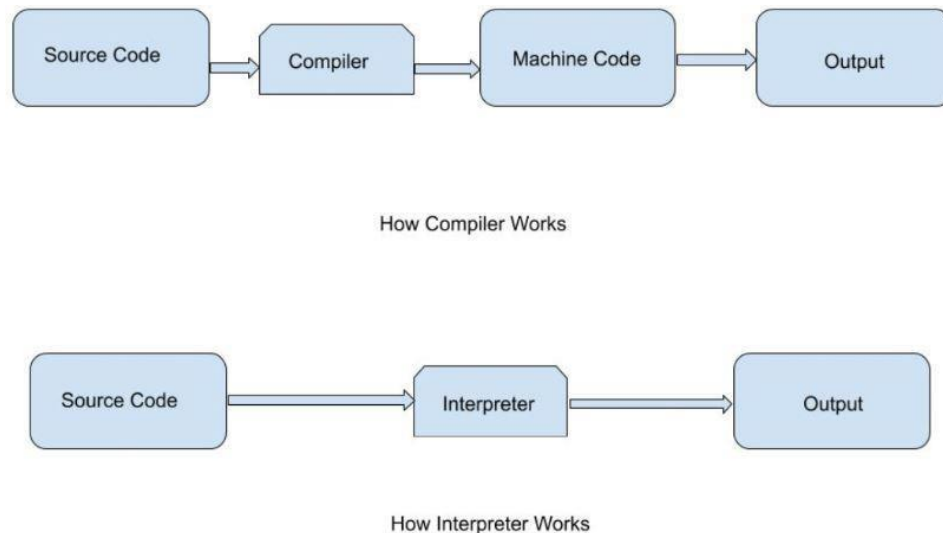


Fig. 1.1 Comparison of the work of the compiler and the interpreter [4]

### 1.1.1 Python and big data

If we are considering choosing a programming language for developing a program using big data, *Python* is a good choice. It allows developers to write a program with fewer lines compared to lower-level languages. Importantly, however, it contains many advanced and useful libraries for scientific calculations [5].

***Python* is a suitable tool for data analysis and working with big data due to its following features:**

- **Open-source:**

*Python* is an *open-source* programming language. It is not owned by anyone, is easily accessible, and is maintained by a community of programmers. This means that people can edit it. *Python* can be used by both Windows and Linux users [5], [6].

- **Library Support:**

*Python* is widely used for scientific computing in both academic and industrial fields. It consists of a large number of tested analytical libraries, which include packages for numerical calculations, data analysis, statistical analysis, visualization, machine learning [5].

**Some of the popular *Python* libraries used in the data domain:** *Pandas*, *Numpy*, *Scikit-learn*, *SciPy*, *Mlpy*, *Matplotlib*, *TensorFlow*, *Theano*, *NetworkIX*, *SymPy*, *Dask* [5].

- **Velocity:**

*Python* has a fast data processing speed. Its codes are executed in a fraction of the time that other programming languages take, thanks to its simple syntax and easy-to-manage code [7].
- **Range:**

*Python* is an object-oriented programming language that supports higher-level data structures such as *lists*, *sets*, *tuples*, *dictionaries*, and others. It supports many scientific operations, such as working with matrices, with data frames. The given capabilities of *Python* extend the scope to simplify and speed up data operations, which makes its use with big data a very powerful combination [5], [7].
- **Support for data processing:**

*Python* has a built-in data processing support feature for unconventional and unstructured data (image, voice data), which is the most common requirement for social media big data analysis. This feature is one of the reasons why large data companies choose *Python* as a basic requirement for working with big data [5], [7].
- **Easy to write a program:**

Compared to other programming languages, programming in *Python* is simple. We can run multi-line programs and can quickly connect and identify data types [7].
- **Easy to learn:**

*Python* is simple to learn due to its simple syntax. It helps big data professionals to pursue knowledge in their sphere instead of wasting time understanding the technicalities of a programming language [7].
- **High Hadoop Compatibility:**

Both *Python* and *Hadoop* are *open-source* big data platforms, so they are more compatible with each other than in combination with any other programming language. Therefore, developers usually choose *Python* for working with *Hadoop* due to its extensive support for libraries. *Python* includes the *PyDoop* package, which offers excellent support for *Hadoop* [7].
- **Portability:**

Many operations are performed in *Python* simply because of its accuracy and extensibility [7].

- **Large Community Support:**  
Big data analytics typically deals with complicated problems that require community support to solve them. *Python* has a large and active community support that helps data scientists and programmers with the problems that arise [7].
- **Scalability:**  
When working with data, scalability plays an important role. *It* makes Python faster compared to other programming languages. As the volume of data increases, *Python* will increase the speed of its processing [7].

### 1.1.2 Python libraries used in the field of data analysis

In the next section, we will look at selected libraries of the Python programming language that are used in the data domain and were already listed in the previous subchapter.

#### 1.1.2.1 Pandas

*Pandas* stands for *Data Panel* or *Python Data Analysis*. It is an *open-source* library designed for simple and intuitive work with relational or labeled data. These are data that we can classify based on the label assigned to them. *Pandas* provides various data structures and operations to manipulate numerical data and time series. The said library is built on the *NumPy* library [8], [9], [10].

**The Pandas library is characterized by the following advantageous features [8]:**

- enables fast and efficient data handling and analysis
- allows you to easily retrieve data from various file objects
- allows you to flexibly transform and pivot datasets
- Provides a time series feature

Note: Pivoting is a process in which the data in a table is transformed from a height arrangement to a wide arrangement (see Figure 1.2). The data is divided into columns, which usually aggregate the values [11].

Store	Product	Sales
A	TV	2
A	TV	4
B	TV	6
B	DVD	8

Store	Avg(Sales) for TV	Avg(Sales) for DVD
A	3	(Empty)
B	6	8

Fig. 1.2 Pivoting process [11]

**What can be done with *the Pandas* library [8], [10]:**

- Clean, merge, and join datasets
- easy manipulation of missing data (represented as NaN)
- it is possible to insert and delete columns from the DataFrame data structure and higher-dimensional objects
- Visualize data
- Determine the correlation between two or more columns of a data structure
- find out the average value, minimum, maximum

The basic data structures of *Pandas* include *Series* and *DataFrame*. *Series* is a one-dimensional array capable of storing data of any type. A *DataFrame* is a two-dimensional, heterogeneous, resizable table structure with labeled axes – rows and columns [8].

### 1.1.2.2 NumPy

*NumPy* (*Numerical Python*) is an *open-source* library used in almost all areas of science and technology. It is a universal standard for working with numerical data in *Python*. The *NumPy* API (*Application Programming Interface*) is widely used in *Pandas*, *SciPy*, *Matplotlib*, *scikit-learn*, *scikit-image*, and many others used in the field of data science [12].

*NumPy* contains multidimensional data structures of arrays and matrices. It provides a homogeneous n-dimensional object of the *ndarray* array with methods to work with it. It adds powerful data structures to *Python* that ensure efficient computations with arrays and matrices, and adds a number of high-level mathematical functions working with given arrays and matrices. In addition to those mentioned, *Numpy* includes linear algebra functions, Fourier transform, pseudorandom number generation [12], [13].

*Python* allows you to create *lists* without *Numpy*, but *Numpy* improves the performance of operations. Its arrays are *memory arrays*.

efficiently, as they store homogeneous data in contiguous blocks of memory, where heterogeneous data can also be stored in lists. *NumPy* supports multidimensional arrays and provides features for working with them, such as *broadcasting*. It allows you to perform operations between arrays of different shapes (see Figure 1.3) [13].

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([[5], [10], [15]])
result = arr1 + arr2 # [[6, 7, 8], [11, 12, 13], [16, 17, 18]]
```

Fig. 1.3 *Broadcasting* two *NumPy* arrays of different shapes [13]

### 1.1.2.3 Scikit-learn

*Scikit-learn* is an *open-source* machine learning library built on top of *NumPy*, *SciPy*, *Matplotlib*. It contains simple and effective tools for predictive data analysis. It supports both controlled and uncontrolled learning [14], [15].

**The Scikit-learn *library* contains functions for working in the following areas of machine learning:**

- **Classification** [14]:

Identifies which category the object belongs to.

Applications: spam detection, image recognition

Algorithms: *gradient boosting*, nearest neighbor method, random forest, logistic regression, and others

Figures 1.4 and 1.5 show the results of the classification of points belonging to two classes using different algorithms. The respective columns represent, in order from the left in Figure 1.4, the input data, the nearest neighbor method, the linear SVM (*Support Vector Machine*), the RBF (*Radial Basis Function*) SVM, Gaussian processes and decision trees. In Figure 1.5, the corresponding columns, in order from left, represent a random forest, a neural network, *AdaBoost*, a naïve Bayesian classifier, and QDA (*Quadratic Discriminant Analysis*).

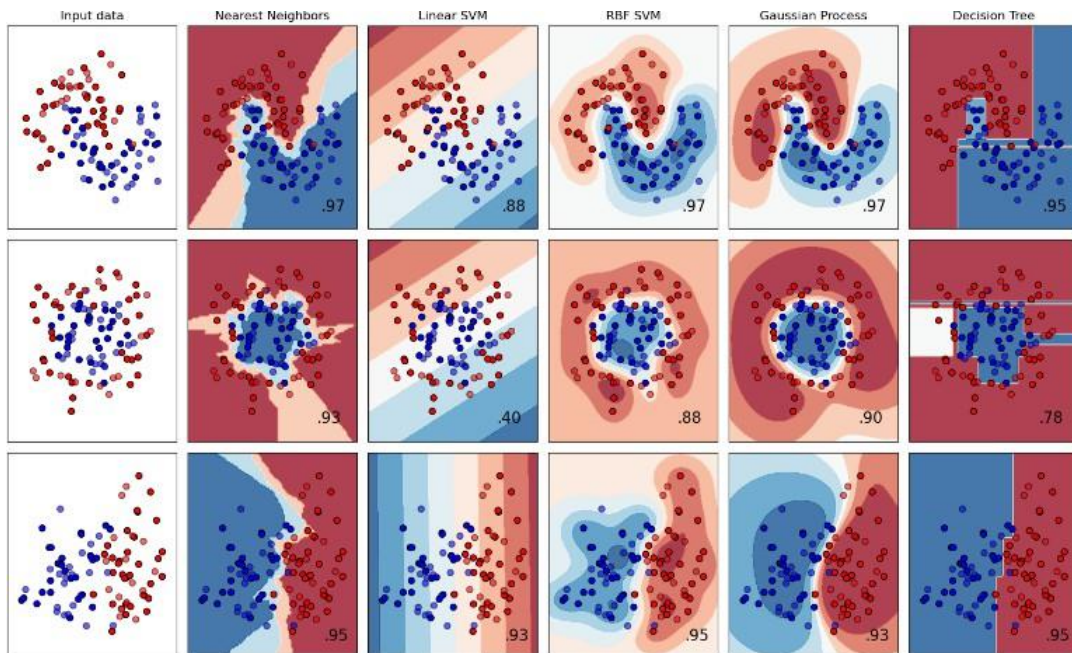


Fig. 1.4 Comparison of properties of different classifiers, part a [16]

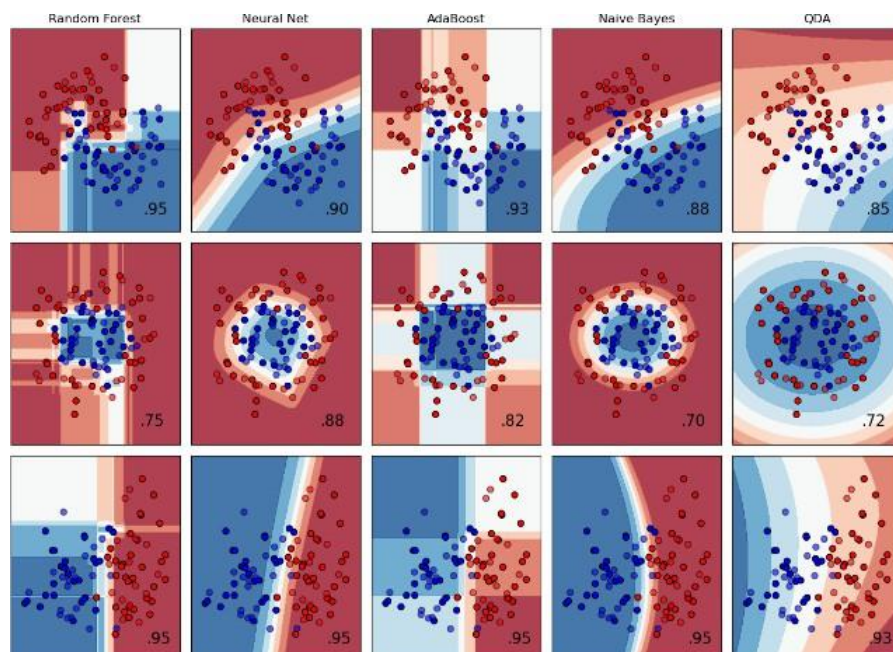


Fig. 1.5 Comparison of properties of different classifiers, part b [16]

- **Regression** [14], [17]:

It is a method for investigating the relationship between independent variables and dependent variable. It is used to predict the values of a continuous attribute associated with an object.

Applications: drug response, stock prices



Algorithms: *gradient boosting*, nearest neighbor method, random forest, ridge regression, and others

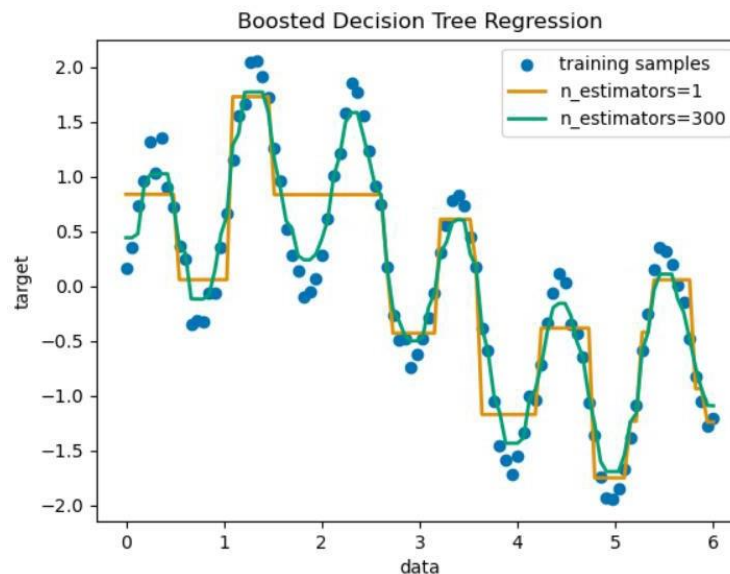


Fig. 1.6 Regression result using the decision tree method and *AdaBoost* [18]

- **Clustering** [14]:

Automatic grouping of similar objects into a single set.

Apps: Customer grouping

Algorithms: *k-means*, HDBSCAN (*Hierarchical Density-Based Spatial Clustering of Applications with Noise*), hierarchical clustering, and others

Figures 1.7 and 1.8 show the results of point clustering in several possible arrangements using different algorithms. The respective columns represent, in order from left in Figure 1.7, *the k-means methods*, affinity propagation, *meanshift*, spectral clustering, *ward*, and agglomerative clustering. In Figure 1.8, the respective columns represent DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*), HDBSCAN, OPTICS (*Ordering points to identify the clustering structure*), MIRCH and Gaussian mixture in order from left.



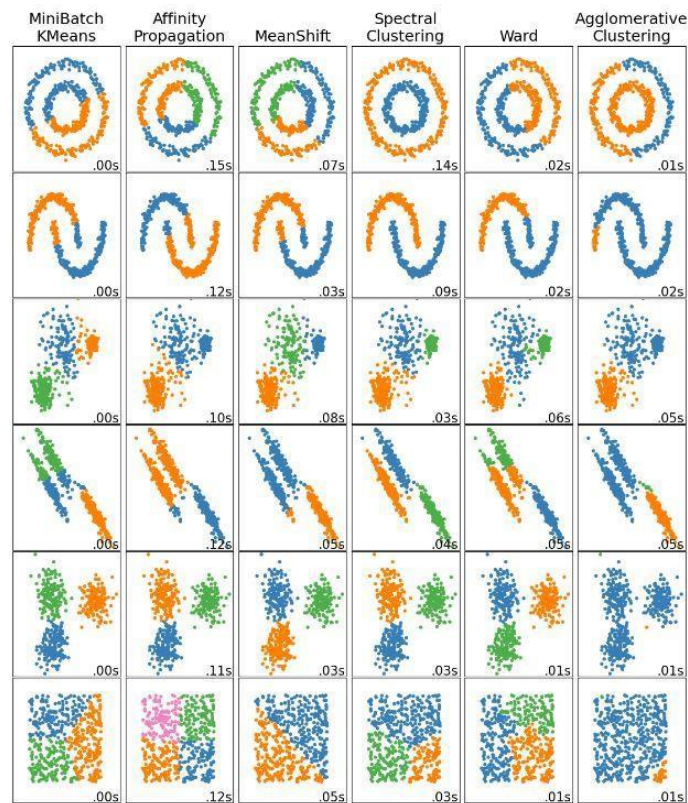


Fig. 1.7 Comparison of properties of different clustering algorithms, part a [19]

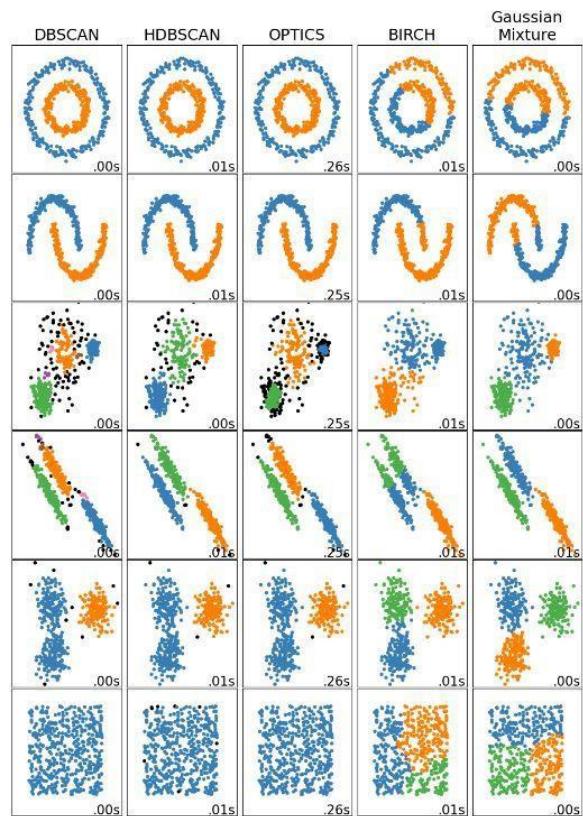


Fig. 1.8 Comparison of properties of different clustering algorithms, part b [19]

- **Reducing the size of the data** [14], [20]:

This is a technique used to reduce the number of symptoms in a dataset while retaining as much important information as possible.

Applications: visualization, increased efficiency

Algorithms: PCA (*Principal Component Analysis*), Feature Selection, Non-Negative Matrix Factorization, and Others

- **Model selection** [14]:

Comparing, verifying and selecting parameters and models.

Applications: increased accuracy due to parameter modification

Algorithms: grid search, cross-validation and others

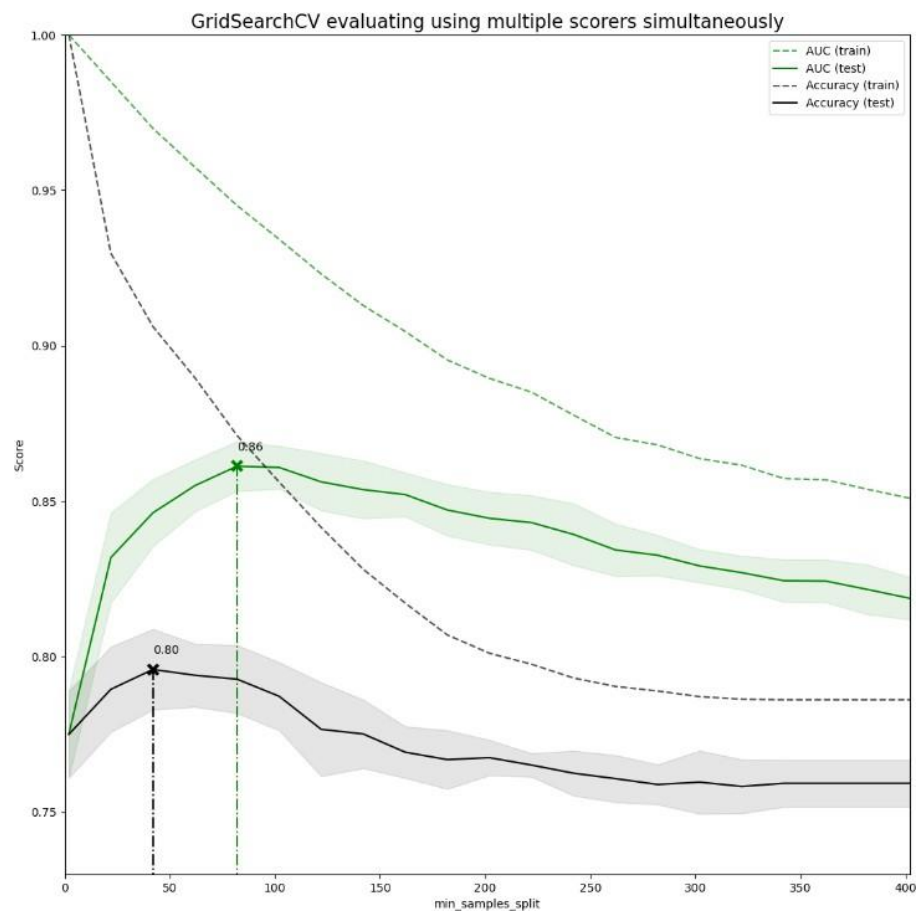


Fig. 1.9 Comparison of model accuracy on training and test data [21]

- **Pre-processing** [14]:

It is about the extraction of symptoms, the normalization of data.

Applications: Transform input data (such as text) for use with machine learning algorithms

Algorithms: data preprocessing, symptom extraction, and more

#### 1.1.2.4 SciPy

*SciPy (Scientific Python)* is an *open-source* library developed and maintained by a community of programmers. It contains basic algorithms for scientific calculations in *Python*. *SciPy* provides algorithms for optimization, integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics, and many more. It extends the *NumPy* library with additional tools for counting arrays and provides specialized data structures such as sparse matrices and k-dimensional trees [22].

#### 1.1.2.5 Mlpy

*MLPY (Machine Learning Python)* is another of the *open-source* libraries used to use machine learning methods for controlled and uncontrolled learning. It is aimed at finding a reasonable compromise between modularity, maintainability, reproducibility, usability and efficiency. *Mlpy* is built on the *NumPy*, *SciPy*, and GNU (*GNU's Not Unix*) scientific libraries [23].

#### **Possibilities of working with *the Mlpy* module and algorithms related to the given areas** [23]:

- **Regression:**

least squares method, ridge regression, last angle regression, elastic meshes, nuclear ridge regression, vector machine support (SVM), partial least squares method

- **Classification:**

linear discriminant analysis (LDA – *Linear Discriminant Analysis*), basic perceptron, elastic network, logistic regression, vector machine support (SVM), diagonal linear discriminant analysis (DLDA – *Diagonal LDA*), Golub classifier, based on *the Parzen* method, *Fisher* discriminant classifier k-nearest neighbors, iterative RELIEF, classification tree, maximum probability classifier

- **Clustering:**  
Hierarchical clustering, memory-saving hierarchical clustering, *k-means*
- **Reducing the size of data:**  
Fisher's Discriminant Analysis (*FDA*), Spectral Regression Discriminant Analysis (*SRDA*), Principal Component Analysis (*PCA*)
- **Wavelet submodule:**  
discrete and continuous wave transform
- **Different:**  
evaluation algorithms, function selection, Canberra stability indicator, oversampling algorithms, error evaluation, vertex search algorithms, *Dynamic Time Warping (DTW)*, Longest Common Subsequence (*LCS*)

### 1.1.2.6 Matplotlib

*Matplotlib* is a comprehensive library for creating static, animated, and interactive visualizations. It makes easy things simple and difficult things possible. *Matplotlib* comes with a wide range of graphical representations, such as line, bar chart, *scatter chart*, histogram, and more. Charts help you understand trends and patterns and allow you to track correlations between variables. They are tools for thinking about quantitative information [24], [25].

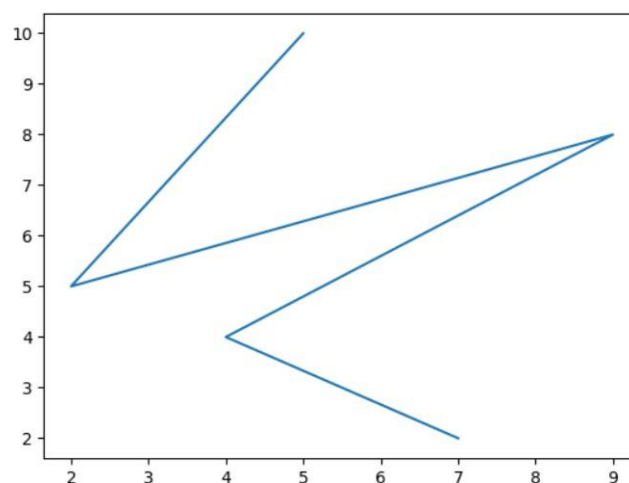


Fig. 1.10 Plotting a line using the `plot()` function [25]

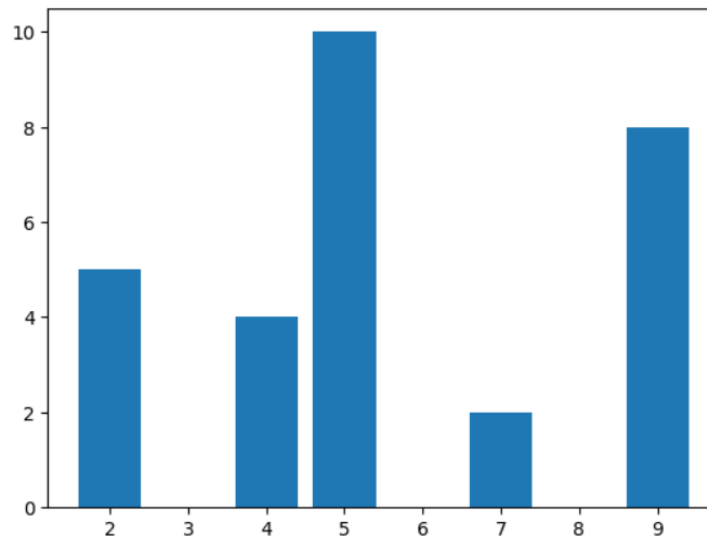


Fig. 1.11 Plotting a bar graph using the `bar()` function [25]

### 1.1.2.7 TensorFlow

*TensorFlow* is an *open-source* library making it easy for beginners and experts alike to build machine learning models for desktop, mobile, web, and *cloud*. It was originally developed for the purpose of conducting machine learning and deep neural network research [26], [27].

**In machine learning, data plays a key role. *TensorFlow* offers multiple data tools to consolidate, cleanse, and preprocess data at scale [26]:**

- Standard datasets for initial training and validation
- Highly scalable *data pipelines* for data retrieval
- Preprocessing layers for common input transformations
- Tools for validating and transforming large datasets

### 1.1.2.8 Theano

*Theano* is a library that allows efficient evaluation of mathematical operations, including operations with multivariate arrays. It achieves high speeds, and works much faster on a graphics processing unit (GPU – *Graphics Processing Unit*) than on a CPU (*Central Processing Unit*). *Theano* is mainly designed to handle the computations required for neural network algorithms used in deep learning. For this reason, it is a very popular library in the field of deep learning [28].

### 1.1.2.9 NetworkX

*NetworkX* is a package for creating, manipulating, and studying the structure, dynamics, and functions of complex networks. It is used to study large complex networks represented in the form of graphs with nodes and edges. With *NetworkX*, we can generate many types of random and classical networks, analyze the network structure, build network models, design new network algorithms, and draw meshes [29].

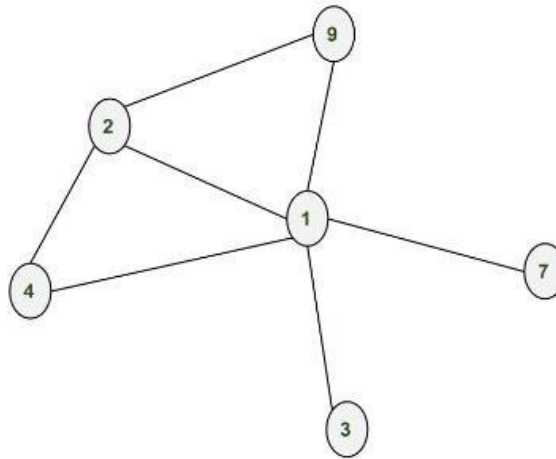


Fig. 1.12 Graph showing a network using nodes and edges between them [29]

### 1.1.2.10 SymPy

*SymPy* (*Symbolic Python*) is a library for symbolic mathematics. Its goal is to become a full-fledged computer algebra system while keeping the code as simple as possible. The effort is to make the code understandable and easy to extend. *SymPy* is written entirely in *Python*. It only depends on the pure Python library *mpmath*, which works with floating-point arithmetic [30].

```
# import everything from sympy module
from sympy import * x = Symbol('x')
y = Symbol('y')

z = (x + y) + (x-y)
print("value of z is :" + str(z))
```

Fig. 1.13 Using the `Symbol()` function to create symbolic variables [30]

```
value of z is :2*x
```

Fig. 1.14 Output from the example shown in Figure 1.13 [30]

### 1.1.2.11 Dask

*Dask* is a library that supports parallel computing in *Python*. It provides features such as dynamic task scheduling, which is optimized for interactive computing workloads[31].

In most cases, the *Pandas* and *NumPy* libraries are used in big data analysis. The aforementioned packages support a number of calculations. However, when the data set does not fit into memory, the packets do not change accordingly. If the dataset does not fit into memory, *dask* expands the dataset to fit on disk. *Dask* allows scaling to *clusters* (clusters) or scaling down to a single machine according to the size of the dataset [31].

```
CPU times: user 619 ms, sys: 73.6 ms, total: 692 ms  
Wall time: 705 ms
```

Fig. 1.15 Reading the selected dataset using the `read_csv()` function of the library *Pandas* [31]

```
CPU times: user 21.7 ms, sys: 938  $\mu$ s, total: 22.7 ms  
Wall time: 23.2 ms
```

Fig. 1.16 Reading the same dataset using the `read_csv()` function of the library *dask* [31]

## 1.2 Scala

*Scala* (*Scalable language*) is a high-level, general-purpose programming language. It is a purely object-oriented language with the possibility of a functional approach. Each value in the *Scala* represents an object that is displayed using a class and its properties (*traits*). Each function in *Scala* represents a value that is also an object. It supports higher-order functions, anonymous functions, as well as nested functions. This programming language is not based on the concept of primitive data (*integer, float, double, char, string, bool*), which can be used to build abstract data types and data structures. *Scala* applies static typing. Data type validation occurs at compile time. Unlike other static programming languages (e.g. *C, C++*), *Scala* does not expect redundant information about the data type from the user. The user does not need to specify it. The compiler can deduce at compile time what type it is [32], [33], [34].

*Scala* works with *Java*. It is designed to work with *the Java Runtime Environment*. Its programs are converted to a *.class*, which contains *the Java Byte Code*. They can then run on a *Java Virtual Machine*. In addition to the above platform, *Scala* also works on *JavaScript and LLVM* (Low Level Virtual Machine) platforms [32], [35].

*Scala* is a popular programming language for many reasons. As it is a high-level language, it is easy to learn. It contains the functions of several programming languages, such as *C, C++, Java*, which makes *Scala* more scalable and productive. It is used to create web applications, where it supports compiling the program into *JavaScript* and also allows you to create desktop applications, where the program is compiled into *Java Virtual Machine Byte Code*. Due to its complexity, *the Scala* achieves higher efficiency in performance [32].

Just as the combination of object-oriented and functional approaches makes *Scala* a powerful programming language, it can also be a disadvantage. Two approaches can make it difficult for programmers to understand a given language. Furthermore, compared to the number of *Java* developers, there are only a limited number of *Scala* developers available on the market [32].

*Scala* is mainly used in the field of data analysis in the *Apache Spark* environment. As mentioned earlier, it is further used to create web applications. It provides a tool for creating frameworks and libraries. It allows parallel batch processing to be realized [32].



## 1.3 Python vs. Scala

In the areas of data science and big data, *Python* and *Scala* are among the leading programming languages. *Python* is easy to learn, has a clear syntax, cross-platform compatibility, a large community, and thanks to it, a large number of high-quality libraries for data science and machine learning. On the other hand, it is slower due to its dynamic nature. *The Scala* provides great speed, expandability and reusability. However, its shortcomings are the higher difficulty of learning and also the limited number of developers working with the language. Some data scientists prefer *Python*, others *Scala*. *Python* is the language of choice in the areas of natural language processing and machine learning, as *Scala* does not provide as many tools for these industries [36].

### 1.3.1 Performance

*Scala* is more powerful as compared to *Python*. One of the reasons is the nature of static typing in *Scala*, where the compiler already knows every variable or expression at runtime. *Scala* continues to use *the Java Virtual Machine* as well as a huge number of *Java* libraries, which makes it similar in performance. *At the same time*, *Scala* supports *multithreading*, in which tasks within the application are performed independently. The listed features make *Scala* 10 times faster compared to *Python* [37].

### 1.3.2 Error

*Scala* can find program errors during compilation, as it is a static language. *Python*, on the other hand, is a dynamic language. This feature brings greater flexibility, as the type of variable can change during program runtime. On the other hand, it is prone to errors whenever there are changes to the program. The aforementioned errors are difficult to predict. For this reason, the *Scala* has better features in terms of security [36], [38].

### 1.3.3 Project size

*Python* is suitable for smaller to medium-sized projects. Its extensive library support allows developers to develop and deploy applications quickly. *Scala* is suitable for large-scale complex projects that require performance and scalability [38].

### 1.3.4 Hadoop

*Scala* interacts better with *Hadoop services*, which allow large data sets to be processed in a distributed manner across a cluster of computers. *Scala* works

with *Hadoop* through *Java* APIs, while *Python* must use libraries such as *Hadoopy* [36], [39].

### 1.3.5 Apache Spark

*Apache Spark*, an environment used for big data analysis, is written in *Scala*. Knowledge of *the Scala* makes it possible to understand and adjust its operation. Many of the functions being developed have API interfaces first for *Scala* and *Java*, and later for *Python*. *Python* uses the *PySpark* library to work with *Apache Spark* [36], [38].

## 2 AVAILABLE ENVIRONMENT FOR DISTRIBUTED DATA PROCESSING

The area of big data consists in collecting, processing and analyzing data on the size of petaBytes, exaBytes. Due to the ever-increasing amount of data generated, it is necessary for us to be able to carry out these operations in a reasonable time. *Big Data Frameworks* are tools that allow you to process big data quickly, efficiently, while maintaining its security. These are usually *open-source* environments, so they are available to everyone. The aforementioned environments for working with big data were created to help enterprises successfully integrate the big data domain [40], [41].

The computational effort to train machine learning models has been increasing rapidly – up to 10 times every 18 months since 2010. However, the capabilities of the GPU and TPU (*Tensor Processing Unit*) accelerators used do not even double during the given period. For this reason, organizations need to use 5 times more accelerators for the described needs, and the only way to implement it lies in distributed computing [42].

### 2.1 Apache Spark

*Apache Spark* is an environment for fast parallel processing of large-scale data. It works with a logical group of computers (the so-called *cluster*). It is used in production environments to process data from various sources – HDFS (*Hadoop Distributed File System*), Cassandra databases, Amazon S3 storage service, as well as data from external Google Datastore web services. *Apache Spark* is suitable for a variety of developers because it provides high-level APIs for *the Java, Scala, Python, and R programming languages* [40], [43].

*Apache Spark* runs on Windows, Linux, Mac OS. It works on any platform on which a supported version of Java is executable. *Apache Spark* supports the following versions of these programming languages: *Java 8/11/17, Scala 2.12/2.13, Python 3.8+, and R 3.5+*. When using *the Scala API*, it is necessary for applications to use the same version of *Scala* on which *Spark* was compiled [44].

### 2.1.1 Properties

In the given section, we will look at the most significant features of *Apache Spark*:

- **Velocity:**

*Apache Spark* executes very quickly by caching data in multiple parallel operations. It includes so-called *in-memory computing*, which consists of storing data in the memory of RAM (*Random Access Memory*) servers, which allows quick access. It also includes RDD (*Resilient Distributed Dataset*), which saves time and reduces the number of read/write operations on disk. Thanks to these features, *Apache Spark* can achieve up to 100 times faster speed when processing data in memory compared to *the MapReduce Hadoop* system and up to 10 times faster speed when processing on disk [45], [46].

- **Flexibility:**

As mentioned, *Apache Spark* supports multiple programming languages and allows developers to build applications in *Java*, *Python*, *Scala*, and *R*. *PySpark* is used to use *Python* in *Apache Spark*. It is a *Python* API for *Apache Spark*. It helps to connect the RDD of *Apache Spark* with the *Python* programming language. It supports most of the features of *Apache Spark*, such as Spark kernel, Spark SQL (*Structured Query Language*), Spark Flow, Spark MLlib (see chapter 2.1.2 Composition) [46], [47], [48].

- **Real-time data processing:**

*Apache Spark* is capable of processing data flow in real-time. It receives data in mini-batches on which it performs RDD transformations. For comparison, *MapReduce* allows you to process only stored data [45], [46].

- **Support for multiple workloads:**

A *workload* represents the amount of computational resources and time it takes to complete a task or generate its output. It can be any program or application running on a computer. *Apache Spark* allows you to run multiple *workloads* – interactive queries, real-time analytics, machine learning, and graph processing [45], [49].

- **Advanced analytics:**

*Apache Spark* consists of a number of SQL queries, machine learning algorithms, data stream processing, graph processing, and comprehensive analysis [45], [46].

## 2.1.2 Composition

*Apache Spark* consists of the following elements (see Figure 2.1):

- **Spark Core:**

This is the foundation of *the Apache Spark* environment. The kernel is responsible for distributed task transfer, scheduling, memory management, disaster recovery, task monitoring, and I/O functionality (interaction with storage systems). It uses RDD as its base data type. RDD is designed to hide most of the computational complexity from its users. The kernel is revealed to users through APIs for *the Java, Python, Scala* and *R* programming languages. APIs allow users to see the overall processing process in a simple, high-level form [45], [50].

- **Spark MLlib:**

It is a library of machine learning algorithms. Machine learning models can be trained via *Python* and *R* on any *Hadoop* resource, then they can be saved using *the MLlib* library and imported into *pipelines* built on *Java* or *Scala*. *Apache Spark* was designed so that machine learning algorithms could work quickly using interactive computations running in memory. The algorithms of the given library make it possible to realize classification, regression, clustering, collaborative filtering and pattern mining [50].

- **Spark Flow:**

This is a real-time solution, which is described in Chapter 2.1.4.2 Current Mode . A given part of *Apache Spark* receives data in small batches and allows it to be analyzed using the same application code as for batch analysis. This increases productivity for developers [50].

- **Spark SQL:**

It is a tool that provides interactive queries with the ability to work up to 100 times faster compared to *MapReduce* in *Hadoop*. It is used to work with structured data. It supports various data sources – JDBC (*Java Database Connectivity*), ODBC (*Open Database Connectivity*), JSON (*JavaScript Object Notation*), HDFS, Hive, ORC (*Optimized Row Columnar*), Parquet [46], [50].

- **GraphX:**

It is an environment built on *Apache Spark*. Enables extraction, transformation, and load (ETL), exploratory analysis, and iterative computation

Charts. Thanks given Properties Can Users build and transform a graphical data structure [50].

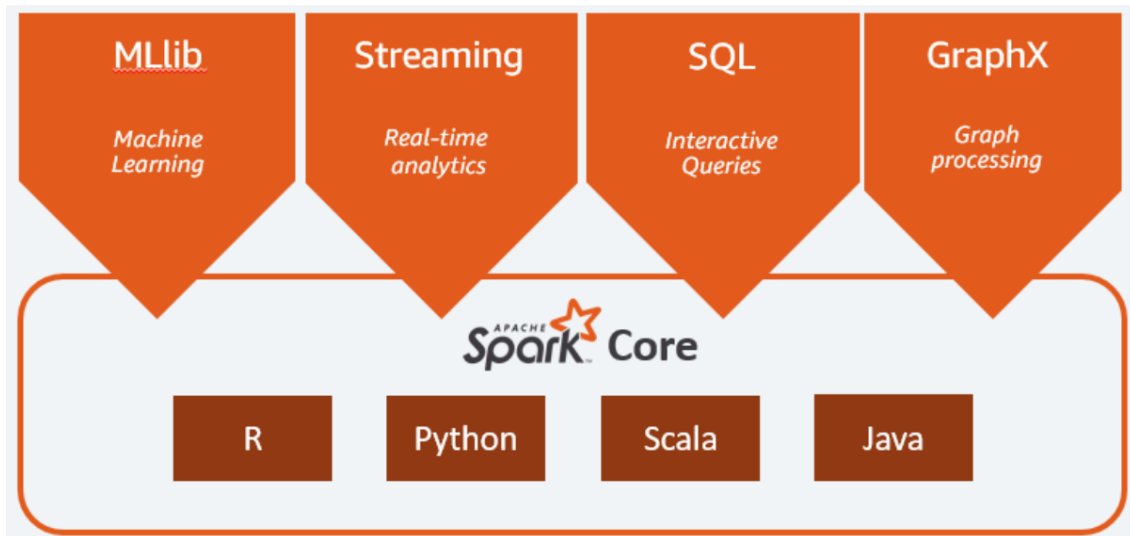


Fig. 2.1 Composition of *Apache Spark* [50]

### 2.1.3 Principle of operation

In the next section, we'll look at how *Apache Spark* works. To understand it better, we also need to describe its basic object, which is RDD.

#### 2.1.3.1 RDD - Resilient Distributed Dataset

RDD is the basic building block of *Apache Spark*. His initials have the following meanings [51]:

**R – Resilient (Durable):**

- fault tolerance, ability to recover data in case of failure

**D – Distributed:**

- Distributing data across multiple nodes in *a cluster*

**D – Dataset:**

- A set of split data with values

Within the RDD, each dataset is divided based on the key into logical partitions (see Figure 2.2), which can be worked with on different nodes within *the cluster*. This makes it possible to perform data operations in parallel. With the strength of multiple nodes, calculations can be performed with the dataset very quickly. In addition, RDD is

capable of quick recovery from problems, as the same parts of the data are processed on several *executor nodes*. In the event that one of the nodes fails, another can provide the desired result instead of it [51].

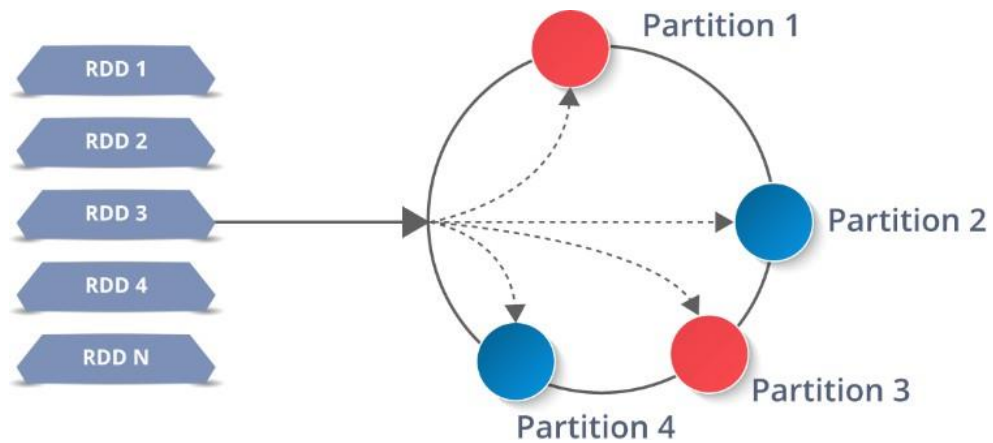


Fig. 2.2 Division of RDD into partitions [51]

RDD is an immutable object. Once it has been created, it cannot be modified. However, it can be used to perform two types of operations – transformations and actions. Transformations are used to create new RDDs. The actions are used *by Apache Spark* to perform calculations, the result of which is then passed *to the driver* (see chapter 2.1.3.2 Apache Spark architecture) [51].

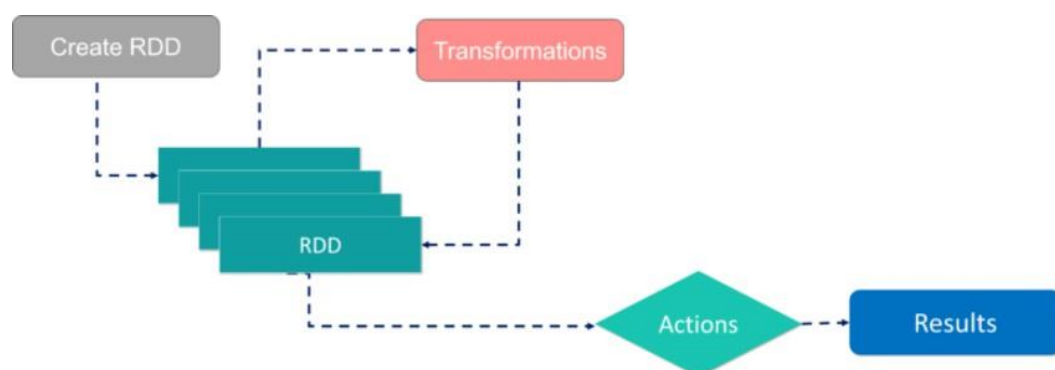


Fig. 2.3 RDD operations [51]

### 2.1.3.2 Apache Spark architecture

Figure 2.4 shows the architecture of *Apache Spark*. It consists of three groups of important components – the *master node*, the *cluster manager*, and the *executor nodes*. Let's describe them in more detail [51].

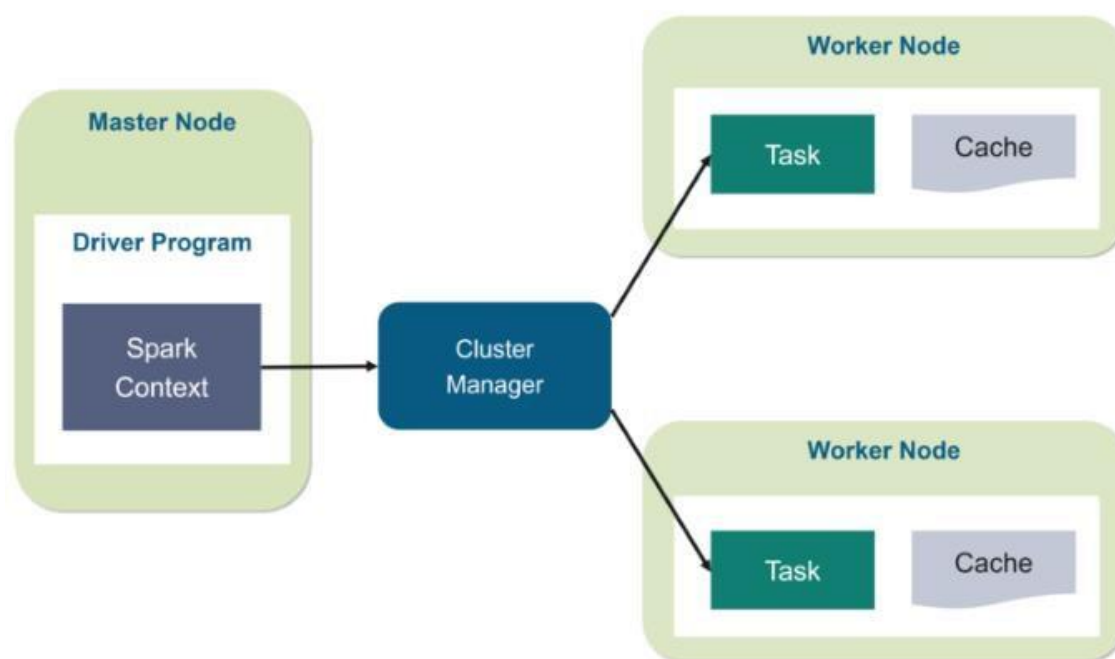


Figure 2.4 *Apache Spark architecture* [51]

The *master* node contains a so-called *driver* program that controls the entire *Apache Spark* application. It ensures the conversion of the user program into tasks. He takes care of the running of the program within *the cluster*. As can be seen in Figure 2.4, the *driver* program contains the so-called *Spark Context*, which provides access to individual functionalities of *Apache Spark* [51], [52].

A *cluster* manager is in charge of assigning tasks to individual execution nodes. The mentioned nodes also have a cache. *Spark Context* distributes RDD objects after they have been created to various execution nodes that can store them within said memory. The *cluster* manager can increase or decrease the number of execution nodes depending on the need for the currently processed data. [51], [52].

Execution nodes represent slave-type elements. They perform the assigned tasks on the appropriate RDD partitions and then return the result of the *Spark Context*. In the case of



With a higher number of nodes listed, tasks can be distributed and performed more efficiently. This will achieve a higher processing speed [51].

## **2.1.4 Work modes**

*Apache Spark* works in two modes – batch and stream [40].

### **2.1.4.1 Batch Mode**

In batch mode, *Apache Spark* reads a large amount of data from a single source and stores the result in memory or on disk. Once the loaded batch has been processed, the selected API (SQL or *DataFrames*) can be used for analysis. This mode of work is suitable if we want to process historical data located in a data repository [40].

### **2.1.4.2 Current Mode**

In stream mode, *Apache Spark* continuously reads incoming data in small batches and then sends it to RDD. It is then possible to apply transformations and actions to the given RDD objects, thus obtaining the results saved in the form of a new RDD object. In this mode, it is not necessary to specify the amount of read data from individual sources. *Apache Spark* can solve the described situation on its own based on the application logic [40].

## **2.1.5 Exploitation**

*Apache Spark* is used in many areas – for example, in the financial sectors, in healthcare, in telecommunications. In the financial sector, it is used to process large amounts of transaction data, to detect fraudulent activities or to assess credit risk with customers. In healthcare, *Apache Spark* is used to review electronic health records, making it possible to better personalize patient care [40].

## 2.2 Ray

Distributed AI systems are too complex and have to face many challenges – for example, data ingestion, pre-processing, training, hyperparameter tuning, providing results. Companies often reach for distributed systems of different brands for individual tasks, each with its own API, data formats and characteristics. To simplify the described issues, *Ray* combines the required tasks into a single environment, which reduces the demands on the development, deployment and management of systems [42].

*Ray* is a unified *open-source* computing environment for distributed computing, allowing you to scale AI (*Artificial Intelligence*) and *Python* workloads. It can parallelize created AI and *Python* applications within a single laptop and then scale them to a physical or cloud *cluster* without the need for code changes [42].

### 2.2.1 Properties

In the given section, we will look at the most significant features of *Ray*.

- **Scalability:**

Users can easily scale their built Python and AI applications using libraries and a few lines of code . They can run their code on their own laptop and then scale it to a *cluster* without the necessary changes [42].

- **Uniformity:**

*Ray* combines various machine learning *workloads*, such as distributed training, hyperparameter tuning, and real-time results. It works with many machine learning projects, such as *Scikit-learn*, *FastAPI*, *XGBoost*, *PyTorch*, *Tensorflow* [42].

- **Openness:**

*Ray* is an environment available to anyone with an active community, [42]

- **Flexibility:**

Being *open-source* and *Python-oriented*, *Ray* is able to integrate machine learning libraries into the applications being developed [42].

- **Portability:**

Applications developed in the *Ray* environment can be executed anywhere – on a laptop, on a physical *cluster*, or on *public* clouds AWS (Amazon Web Services), *GCP* (Google Cloud Platform) and Azure [42].

## 2.2.2 Libraries

*Ray* includes *the Ray AI Runtime* – a suite of the best scalable machine learning libraries. These are the following libraries, thanks to which it is possible to address the above areas of machine learning [42], [53]:

- **Ray Data:** data retrieval, data preprocessing
- **Ray RLlib:** Rewarded Learning
- **Ray Train:** Training Large Models, Deep Learning
- **Ray Tune:** Hyperparameter tuning to find the best model using optimization algorithms
- **Ray Serve:** real-time work – deploying machine learning models

*Ray* takes care of performing distributed computing himself – task scheduling, autoscaling, fault tolerance – allowing researchers to focus on developing application logic instead of studying the principles and control of distributed systems. Through these libraries, even non-experts can use distributed computing [42].

## 3 SERVICES FOR WORKING WITH BIG DATA

As mentioned in the introduction to Chapter 2, Available Environments for Distributed Data Processing, big data is data of enormous size. Many times, this is unstructured data that takes on various forms – for example, it can be discrete data from the production sphere (sensors, IoT (*Internet of Things*) devices), text files with events (log files) or sources with continuous data in data streams with images and audio recordings. As this is an extensive variety of data, it requires specific storage conditions. Data lakes serve this purpose [43].

### 3.1 Data Lakes

A data lake is a place used to store a huge amount of data in its original raw form. It uses so-called object storage, which consists of storing data along with metadata *tags* and unique identifiers that make it possible to access them [54].

**Raw data** is data that has not yet been processed or prepared for specific use in a task compiled by the user. This refers to the processing of received data before storing it in a data lake, as some data sources modify their own data before sending it [55].

**The native format** is the format in which data is sent from a data source [55].

Data lakes are scalable, low-cost, and highly resilient. They allow you to work with data for various analytical purposes in the areas of big data processing, real-time analysis, machine learning, or visualizations. The data can be accessed using various frameworks and analytical tools. Available environments include *Apache Hadoop*, *Presto*, *Apache Spark* [54], [56].

#### 3.1.1 Variety of data stored

Data lakes allow you to store data of any size in different forms from different sources, allowing businesses to have all their data collected in one place without the need for a storage schema. These can be the following types of data [54], [55]:

- **Structured data**

It is organized data with a defined structure, thanks to which it is possible to

The data can be easily stored and accessed using available methods. An example is relational tabular databases [57].

- **Unstructured data**

This is data without a recognizable structure. They represent a collection of data without organization. They are distinguished by their size and heterogeneity. Compared to structured data, unstructured data is difficult to process, understand, and evaluate. Most big data falls into this area. This can be discrete data, textual, image, or audio information [43], [57].

- **Semi-structured data**

It is a combination of the properties of structured and unstructured data. They represent data that does not belong to any database, but contain *tags* that make it possible to distinguish them. An example can be the definitions of the Database Management System (DBMS) tables [57].

### 3.1.2 Data Lake Location

There are several ways to place a data lake:

- **On-site**

Businesses can choose to place a data lake within their own space to maintain security and reliability. This approach provides maximum security (with respect to corporate security) and control. It protects the intellectual property and critical data of the business(55).

- **In the cloud**

*The cloud* allows businesses to take advantage of its various benefits. It provides elastic scalability, faster services, performance, availability, various analytics tools, more frequent updates, and pricing according to the amount and size of services used [55], [56].

- **Hybrid**

The data lake of the enterprise will be divided into two parts – one part will remain in place and the other will be in *the cloud*, where critical data for the business is not stored [55].

Continental uses AWS cloud services for both data storage and processing.

## 3.2 AWS – Amazon Web Services

It is the most widespread and comprehensive *cloud*, providing over 200 fully equipped services. It has a higher variety of functionalities compared to other *cloud* service providers. These include computing capacities, storage, databases, data lakes, machine learning, artificial intelligence and analytical tools. It offers its clients the latest technologies for faster innovation, [58].

AWS has the highest customer community from all over the world. These are millions of people from companies in various industries, such as Coca Cola, Epic Games, Pinterest, Netflix, Philips, Siemens, Toyota, Volkswagen [58].

It is the most secure *cloud* environment today. Thanks to a large set of security tools providing services and functions in the field with the support of 143 security standards, it meets the security requirements of the military industry or global banks [58].

### 3.2.1 S3 – Simple Storage Service

Amazon S3 is Amazon's AWS object storage service. It offers scalability, data availability, security, and performance. Customers across industries can use the service to store and protect any amount of data for use cases such as data lakes, big data, websites, backup and recovery, archiving, IoT devices, mobile and enterprise applications. Amazon S3 offers various functions for storage management, access control, data processing, monitoring, and storage usage analysis [59].

Amazon S3 supports parallel requests, thanks to which the performance of a given service can be scaled. It allows 3,500 requests per second to add data and 5,500 requests per second to retrieve data. Amazon S3 also provides high read-on-write data consistency, thanks to which the request to read an object receives its last modified version [60].

#### 3.2.1.1 Principle of operation

Within Amazon S3, data is stored in objects that represent files and metadata describing those files. When storing data, it is first necessary to create a so-called *bucket* and determine the AWS region. The buckets act as containers for the respective objects. Subsequently, the desired data is uploaded as objects to the created bucket. Each of the objects has a unique identifier in the form of a key, which makes it possible to distinguish it from the others [59].

When creating a bucket, you need to enter its name and specify the AWS region. Subsequently, it is not possible to change the specified parameters. To minimize delays and costs, it is recommended to choose the region that is geographically closest to the customer's location. 100 buckets can be created within a single account, with each bucket containing any number of objects [59], [61].

Objects can be up to 5 TB in size. The basis of each object is a key representing the name of the object and a value containing the stored data. The value can be any sequence of Bytes. Subsequently, the object contains metadata – a name-value pair that describes it. Amazon S3 offers the *S3 Versioning feature*, thanks to which different versions of an object can be stored in a bucket. In this case, it is assigned to each object ID (*Identifier*) of the version. By default, Amazon S3 buckets are private. However, it is possible to change permissions thanks to the *Access Control List (ACL)* [59], [62].

The uniqueness of an object is made up of the bucket name, the name of the object key, and possibly the version ID. If we had an object named `photos/puppy.jpg` (representing the object key) stored in a bucket named `DOC-EXAMPLE-BUCKET` in the US West (Oregon) region, then it can be addressed as follows [59]:

`https://DOC-EXAMPLE-BUCKET.s3.us-west-2.amazonaws.com/photos/puppy.jpg`

### 3.2.1.2 Accessing Amazon S3

There are several options for accessing Amazon S3 buckets:

- **AWS Control Console:**

It is a web-based user interface for managing AWS and Amazon S3 resources [59].

- **AWS Command Line:**

It is a tool through which it is possible to access the various services of AWS using commands written on the command line. In this way, it is possible to work with Linux, macOS, Windows operating systems using Linux *shells* or the Windows command line. There is also remote access, where it is possible *to run commands on EC2 instances using PuTTY or SSH (Secure Shell)* (see chapter 4.2.1 Instances) [63].

Example: Listing bucket objects using the Linux `ls` command:

```
$ aws s3 ls s3://mybucket
      LastWriteTime         Length Name
      -----
                PRE myfolder/
2013-09-03 10:00:00      1234 myfile.txt
```

Fig. 3.1 Listing of *mybucket* bucket objects [64]

- **AWS Software Development Kits (SDKs – *Software Development Kits*):**  
AWS Provides SDKs Consisting from Libraries and codes different programming languages and platforms (*Java, Python, Ruby, .NET, ios, Android*), thanks to which it is possible to access the Amazon S3 [59].
- **Amazon S3 REST API:**  
It is an HTTP (*Hypertext Transfer Protocol*) interface that uses standard HTTP requests to create, retrieve, and delete S3 buckets and objects [59].

### 3.2.2 EC2 – Elastic Compute Cloud

Elastic *Cloud* Computing (EC2) is a web-based service that provides scalable-scale computing capacity in *AWS's cloud*. Thanks to EC2, clients can run a selected number of instances in the form of virtual servers, configure security and network settings, as well as manage storage. Depending on the application, it is possible to add the number of instances to increase or decrease the number of instances so that customers do not have to pay for services they no longer need. There is an upper limit of 20 instances that can be used by a single AWS account [65], [66], [67].

#### 3.2.2.1 EC2 Instances

An instance is a virtual server in the *cloud* that runs on the Linux operating system. There are different kinds of instances that specify the hardware of a guest computer. Each instance offers different computational and memory properties [68].

The creation of instances is based on AMI (*Amazon Machine Image*), which is a template with software configuration (for example, it can contain the type of operating system, application server). Instances are started from AMI, as shown in the image 3.2 [68].



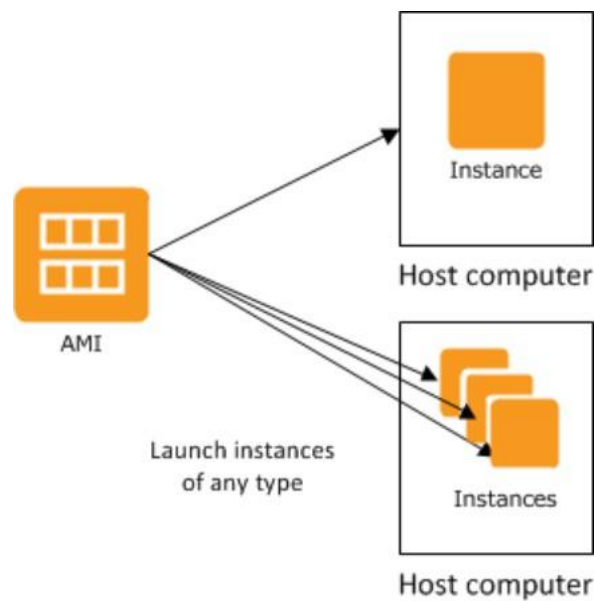


Fig. 3.2 Starting instances from AMI [68]

### 3.2.2.1.1 Types of instances

Each instance type differs from the others by the different size of the parameters of computing capacity, memory size and storage. Based on the above properties, it is included in the relevant family of instances [69].

EC2 assigns some of the resources of the host computer, such as CPU, memory, or instance storage, to a specific instance. It shares the remaining resources with all instances. It is a network or disk subsystem. Each instance type provides different performance from shared sources, and EC2 assigns it the appropriate size accordingly. For example, instance types with high I/O (*Input/Output*) performance allocate a larger volume of shared resources [69].

The nomenclature of instances (see Figure 3.3) consists of a set of symbols that represent a family of instances, an instance generation, a family of processors, additional properties, followed by a period and then a word indicating the size of the instance. Each type of instance comes in several versions differing in the mentioned size. A more detailed description of the individual symbols is given in Table 3.1 [69].

Table 3.1 Meaning of EC2 instance nomenclature [69]

Family of instances		Family Processors		Additional features	
<b>C</b>	Computationally optimized	<b>and</b>	AMD	<b>b</b>	EBS ( <i>Elastic Block Store</i> ) optimized
<b>D</b>	dense fit	<b>g</b>	AWS Graviton	<b>d</b>	Storage instances A large number of
<b>F</b>	FPGA ( <i>Field-programmable Gate Array</i> )	<b>i</b>	Intel	<b>n</b>	Network and EBS Optimized
<b>G</b>	graphically demanding			<b>e</b>	extra storage or memory
<b>HPC</b>	High-performance computing			<b>from</b>	High performance
<b>I</b>	Storage-optimized			<b>q</b>	Qualcomm Inference Accelerators
<b>The m</b>	storage optimized with a vCPU ( <i>virtual centralized processing unit</i> ) ratio to memories one to four			<b>Flex</b>	Flex Instance
<b>Is</b>	storage optimized with a vCPU to memory ratio of one to six				
<b>Inf</b>	AWS inferencia				
<b>M</b>	general use				
<b>Mac</b>	macOS				
<b>P</b>	GPU acceleration				
<b>R</b>	Memory-optimized				
<b>T</b>	Explosive performance				
<b>Thor n</b>	AWS Trainium Accelerator				
<b>U</b>	High memory				
<b>VT</b>	Video transcoding				
<b>X</b>	memory-intensive				

Depending on the type of instance, there are different designs in terms of size – *nano, micro, small, medium, large, xlarge, 2xlarge, 4xlarge, 8xlarge, 12xlarge, 16xlarge, 24xlarge, 32xlarge, 48xlarge*, or versions of metal instances (also in versions of certain multiples) [69].

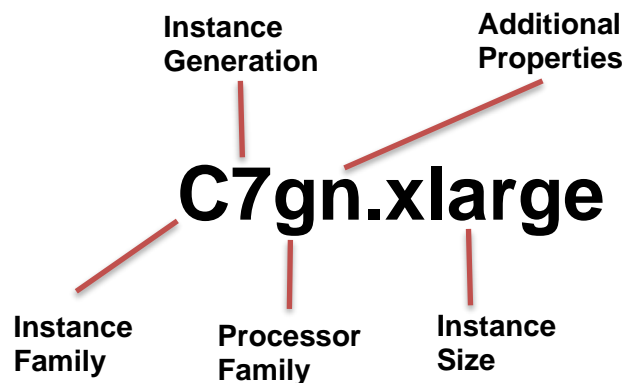


Fig. 3.3 Nomenclature of the EC2 instance [69]

### 3.2.2.1.2 Instance lifecycle

The lifecycle of an instance begins with its launch and ends with termination. Once started, the instance behaves like a guest computer. It can be accessed as a physical computer [68], [70].

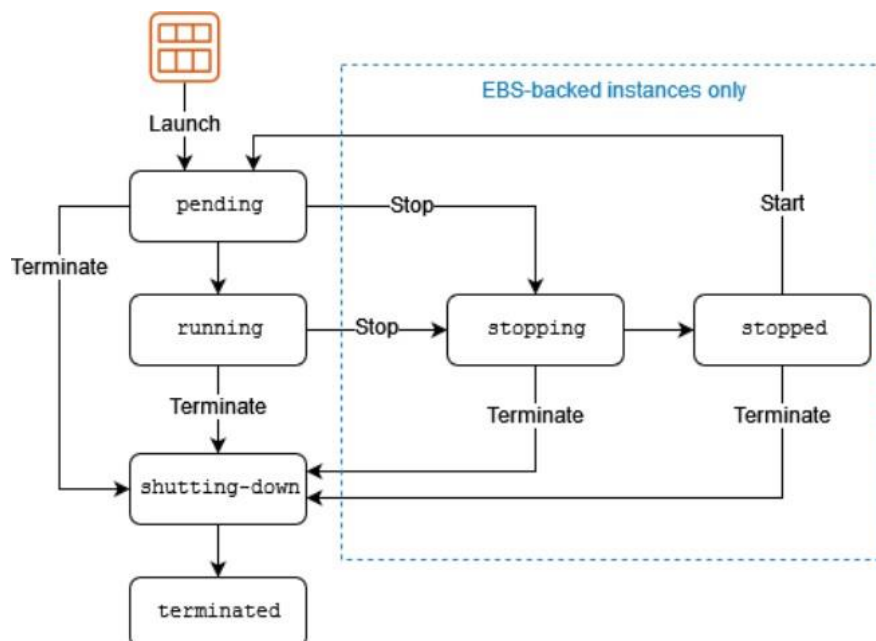


Figure 3.4 Life cycle of an EC2 instance [70]

An instance can be in several states, as shown in Figure 3.4. Once executed, the

instance enters a waiting state *pending*). In this state, the instance is *booted* according to the specified configuration parameters, i.e. according to AMI (see chapter 4.2.1 Instances) [70].

Subsequently, the instance enters the working state (in the diagram in Figure 3.4 marked with the English term *running*). In a given state, instance fees are charged for each second as the only one, with a minimum counting time of one minute [70].

An instance can be turned off from the working state. Then it goes through a *stopping state* until it reaches the stopped off state. From the off state, the instance can then be restarted, which puts it back in *the pending state* [70].

In any of these states, it is possible to terminate the existence of the instance. It is an irreversible process – it is not possible to return to an extinct instance [70].

### 3.2.3 EMR – Elastic MapReduce

Amazon EMR is an AWS manageable *cluster* platform that simplifies the launch of big data environments such as *Apache Spark* or *Apache Hadoop* to process and analyze massive amounts of data. In addition, it allows you to transform and move large amounts of data from (or to some) AWS data storage, such as Amazon S3 [71].

Amazon EMR works with a *cluster*, the principle of which we described in the chapter 2.1.3.2 Apache Spark architecture. When creating a *cluster*, we can choose the environment and applications to be installed within it for the desired use of data processing. In our case, we installed *Spark*, with which we then ran experiments on our dataset [72].

#### 3.2.3.1 Properties

In that section, we'll look at some of the notable features of the EMR platform:

- **Elasticity:**

Amazon EMR allows you to quickly and easily determine the amount of capacity needed, and then either automatically or manually remove the appropriate amount of capacity that is no longer needed (so that users do not have to pay for it). This is an advantageous feature if the customer has variable or unforeseen processing requirements. For example, it is possible that it will need a large amount of computing capacity in the short term. Or it is possible that most of the processing will take place during the night, when it will need a significantly higher number of instances than during the day [73].

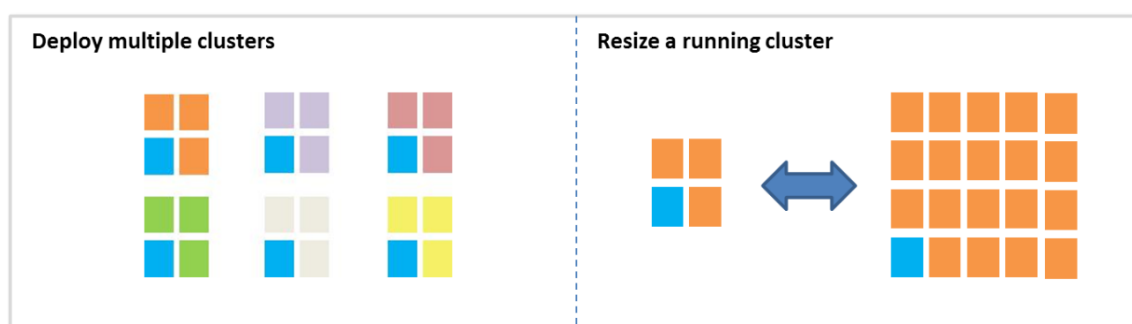


Figure 3.5 Elasticity of computational capacities [73]

There are 2 ways we can change the size of the capacity (see Figure 3.5). The first option (left part of the image) is to create more clusters in case of their necessary use and then, when they have fulfilled their task, it is possible to terminate their existence. It is advantageous to have more *clusters* if we have more users or applications. The second option (right part of the image) is to scale the size of the created *cluster* up or down as needed [73].

- **Flexibility:**

Amazon EMR allows you to work with multiple data storages, such as Amazon S3, HDFS, Amazon DynamoDB [73].

- **Support for various tools for working with big data:**

Amazon EMR supports working with a variety of powerful tools, including *Apache Spark*, *Apache Hive*, *Presto*, and *Apache HBase* [73].

## 4 EXPERIMENTAL WORKPLACE

In order to implement the practical part of the diploma thesis, it was necessary to create materials with which we could carry out experiments on the data of the selected production company of the Continental company. In this chapter, we will look at the creation of an experimental workplace and data processing using AWS services.

### 4.1 Design of an experimental workplace

In the diploma thesis, we implemented the practical part in the Continental data lake, using the services of AWS, which we described in the third chapter. Before we approached the data lake, we formed an idea of what our experimental workplace should contain and how we should apply it in data processing.

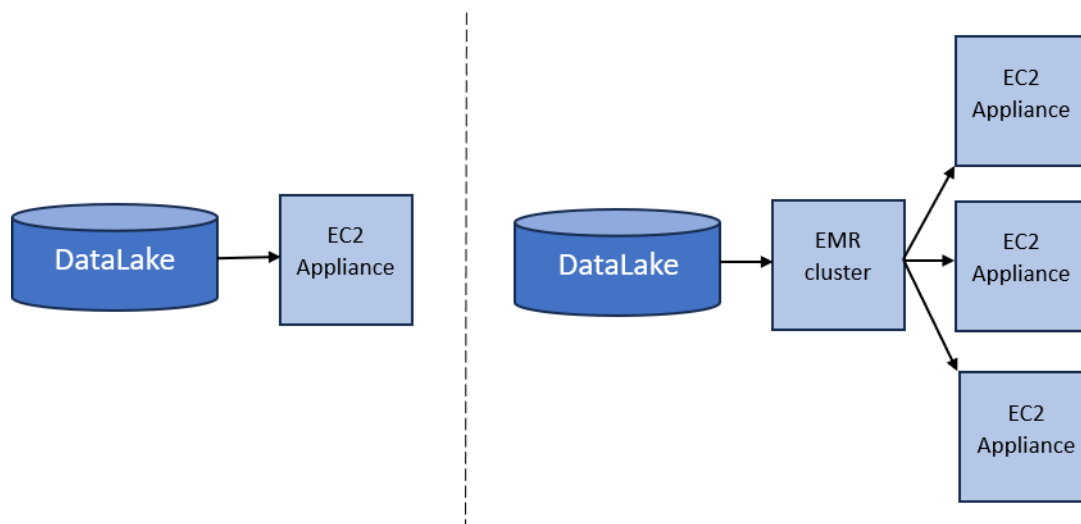


Fig. 4.1 Design of the experimental workplace

Figure 4.1 shows the design of the experimental workplace, which we adhered to when creating it. The main component is a data lake that serves as a repository for the big process data of the Continental manufacturing plant. Subsequently, we needed computing capacities for the purpose of processing selected items of the stored dataset. EC2 instances providing the desired requirements are used for this purpose. When creating the experimental workplace, we decided to first create one EC2 instance and find out what possibilities of work it provides (left part of the picture). Subsequently, we decided to

create an EMR *cluster* based on the *Apache Spark principle*, with which we wanted to process selected data of the selected Continental company (right part of the picture).

## 4.2 Creation of an experimental workplace

After obtaining an AWS account within Continental, it is possible to log in to its data lake. A preview of the lake after the input login (Dashboard part) can be seen in Figure 4.3. We are given a view of all the resources that we use or access within our account.

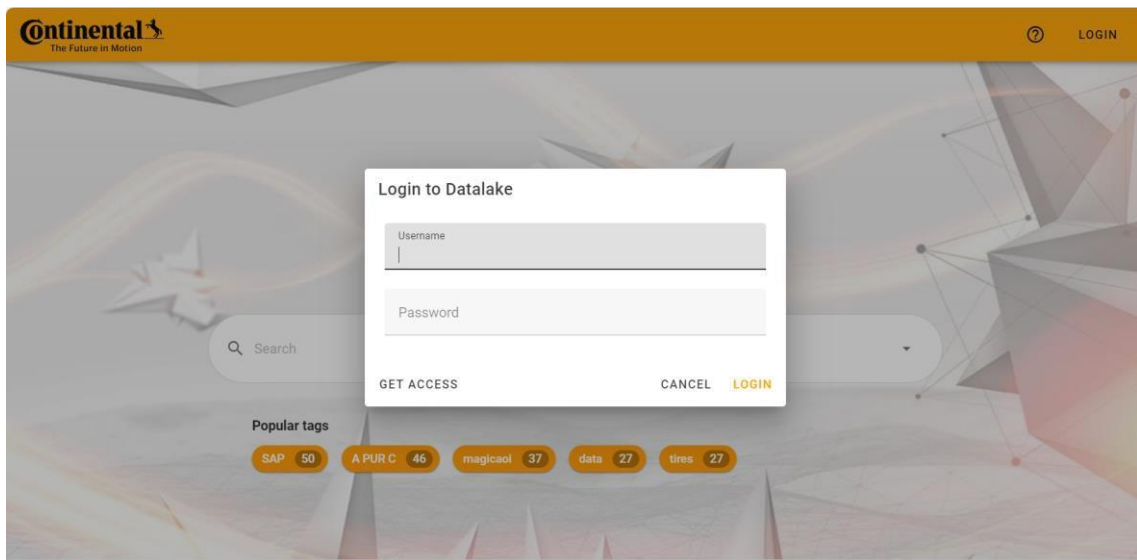


Fig. 4.2 Login window to the Continental data lake

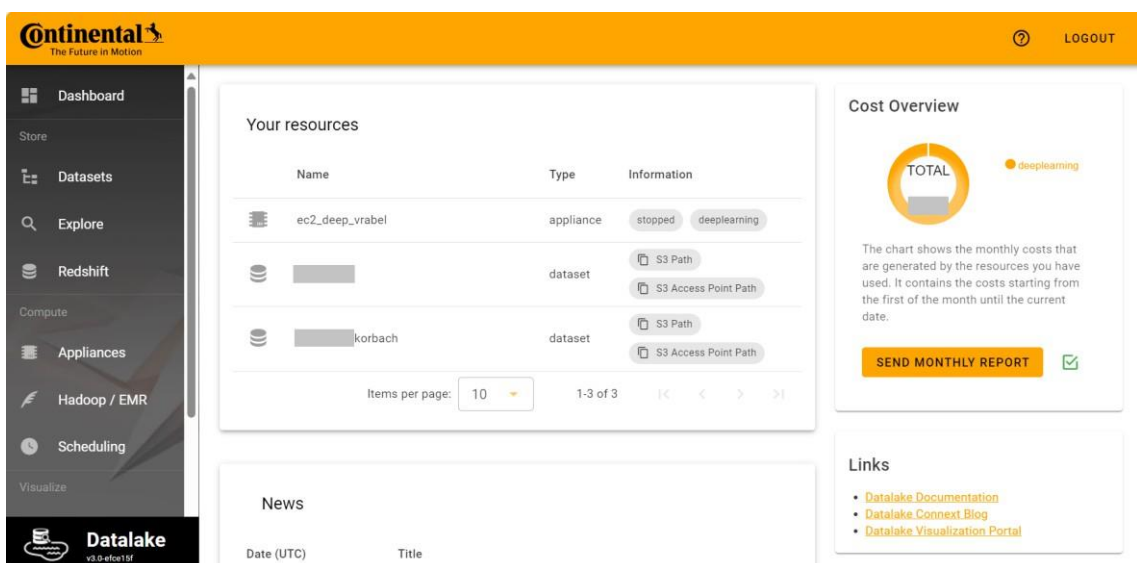


Figure 4.3 Continental's Data Lake

As can be seen in Figure 4.3, at the time of taking the snapshot, we had 3 items available – one EC2 instance named *ec2\_deep\_vrabel* (appliance item type) and two datasets with S3 storage (dataset item type).

Note: For Continental data protection reasons, user names, Internet Protocol (IP) addresses, and other sensitive information are obscured in the company's images.

#### 4.2.1 Instance

In the Continental data lake environment, a new instance can be created in the *Appliances* menu section by clicking the *New appliance* button. As can be seen in Figure 4.4, we have not yet created any instance. After creating it, it will be listed along with its properties.

After clicking on the *New appliance* button, a window with configuration options will appear. This is an AMI selection, based on which the appropriate hardware is assigned to the instance. First of all, it is necessary to choose the area in which we will work with our instance based on the desired application. There are 4 options to choose from (see Figure 4.5):

- **JupyterLab:**

It is a web-based IDE (*Integrated Development Environment*) that allows you to create and share documents containing live code, visualizations, and narrative text. Documents include *Jupyter laptops*, text editors, terminals. It has pre-configured support for *the Python 3* and *Julia programming languages*. *JupyterLab* can be used in data cleaning applications, data transformation, numerical simulation, statistical modeling, machine learning, and data visualization [74].

- **Rstudio:**

It is an IDE specifically designed for the R programming language. *Rstudio* supports data visualization, development of programming packages, and includes a set of robust tools for rendering, history, debugging, and workspace management. The IDE is widely used in data analysis, statistical calculations, and graphical tasks [74].

- **Knime:**

It is an open-source analytical tool that allows you to transform data and perform simple and advanced analyses on datasets. *Knime* provides a user-friendly GUI (*Graphical User Interface*), allowing even less experienced users to perform data analysis [74].



- **Deep Learning:**

It is a deep / machine learning environment (in English) that allows the use of GPUs, thanks to which it is possible to use the given instances for computationally demanding tasks. *Deep Learning* supports the use of libraries or tools in the mentioned area, such as *TensorFlow*, *Apache MXNet* or *PyTorch*. It enables the development and deployment of machine learning models, or experimentation with new algorithms [74].

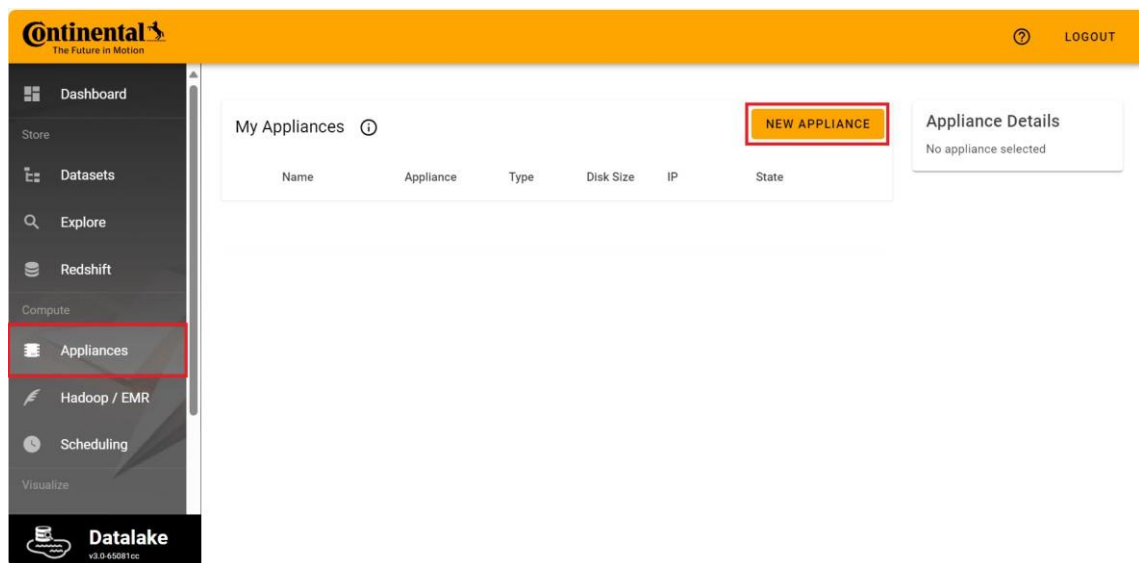


Fig. 4.4 Preview of instances in the Continental data lake environment

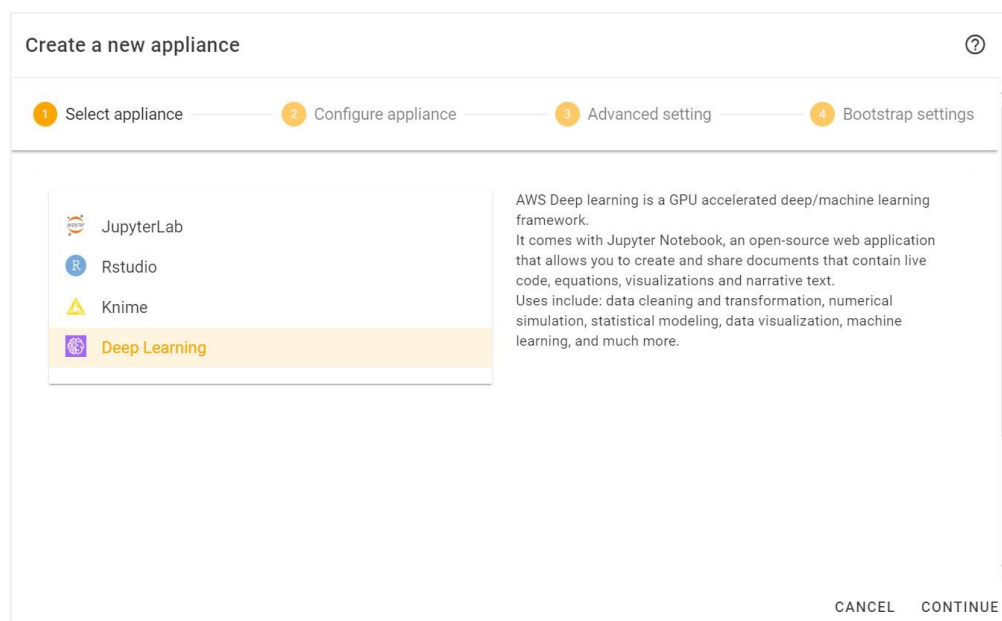


Fig. 4.5 Selection of the area of use of an instance in its creation

When creating the instance, we chose the *Deep Learning* option based on the experience of a master's thesis consultant in the field of big data.

The screenshot shows the 'Create a new appliance' configuration page in the AWS console. The instance name is 'ec2\_deep\_vrabel'. The 'Deep Learning' AMI is selected. The 'Use GPU' toggle is off. The instance type is 't3.xlarge' with 4 CPUs and 16 GB of memory. The disk size is set to 180 GB. The billing information section is greyed out. Navigation buttons 'BACK', 'CANCEL', 'CREATE', and 'CONTINUE' are visible at the bottom.

Fig. 4.6 Instance configuration

When configuring the EC2 instance, we also based on the recommendations of the diploma thesis consultant. As you can see, the instance needs to be named. We chose lowercase letters because of working with the Linux operating system. Subsequently, we chose 4 processor cores, plus 16 GB of memory and 180 GB of disk size. Based on the selected parameters, the instance type *T3.xlarge* was assigned for the given AMI template (see chapter 3.2.2.1.1 Types of instances). It is also worth noting the grey window in which there is information about fees (for the protection of sensitive data, specific values are not given). In the *cloud*, the user only pays for what he uses. Therefore, based on the selected parameters of the instance, its hourly rate is determined. If we chose higher performance or memory, the price would rise. Subsequently, we kept the original settings when we finished creating the instance.

Name	Appliance	Type	Disk Size	IP	State
ec2_deep_vrabel	deeplearning	t3.xlarge	180 GB	[REDACTED]	stopped

ec2_deep_vrabel	
Appliance	deeplearning
Type	t3.xlarge
Disk Size	180 GB
Created	7 days ago
Last updated	7 days ago

Fig. 4.7 Type of EC2 instance created by us

As mentioned in chapter 3.2.2.1.2 Instance Lifecycle, an instance can be in several states. The given states can be changed directly in the instance options (see figure 4.8). We are usually interested in three of them. If we do not use the instance, we will put it in *the stopped* state so that we are not charged for it. To do this, press the *Stop appliance* button. If we want to work with the instance, click on the *Start appliance* button, which will convert it to the working state. As soon as we no longer need to use the services of a given instance, it is possible to terminate its operation from any state by pressing the *Terminate appliance* button.

Name	Appliance	Type	Disk Size	IP	State
ec2_deep_vrabel	deeplearning	t3.xlarge	180 GB	[REDACTED]	stopped

- Open DeepLearning
- Open SSH Console
- Start appliance
- Stop appliance
- Terminate appliance
- Change type

Fig. 4.8 Instance states

In the instance options, in addition to changing the state, it is also possible to open environments in which the user can interact with it directly. To do this, it first needs the instance to be in working condition. The first of the environments is *DeepLearning*

(opens by pressing the *Open DeepLearning button*). It is a graphical environment in which it is possible to write *Python* scripts for a given instance.

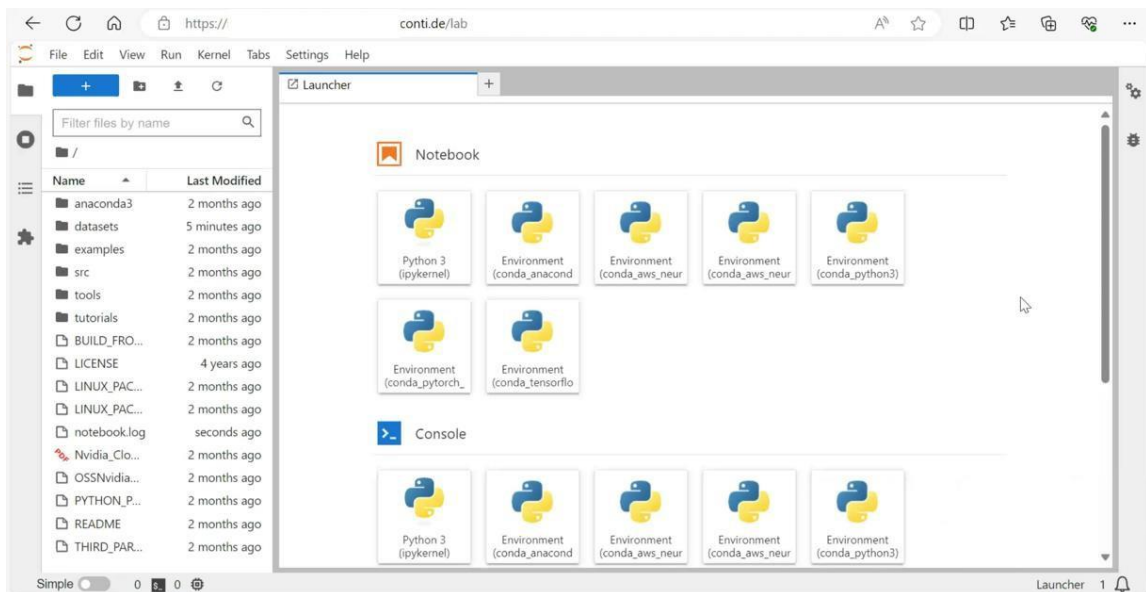


Fig. 4.9 DeepLearning graphical environment

The second environment is the SSH console (opens by pressing the *Open SSH Console* button). The user gives commands to the instance via Linux commands.

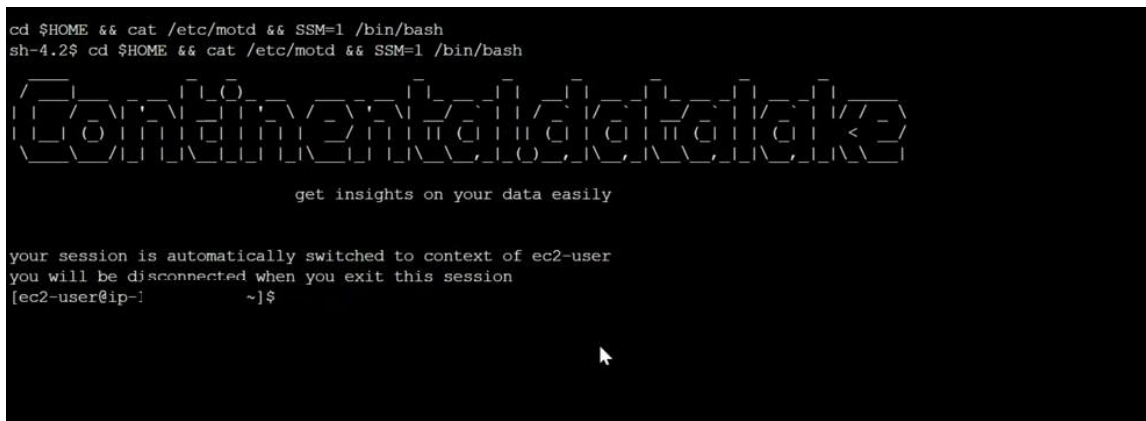


Fig. 4.10 SSH console

## 4.2.2 Dataset

Continental's data lake has storage for data in S3 format (see chapter 3.2.1 S3 – Simple Storage Service).

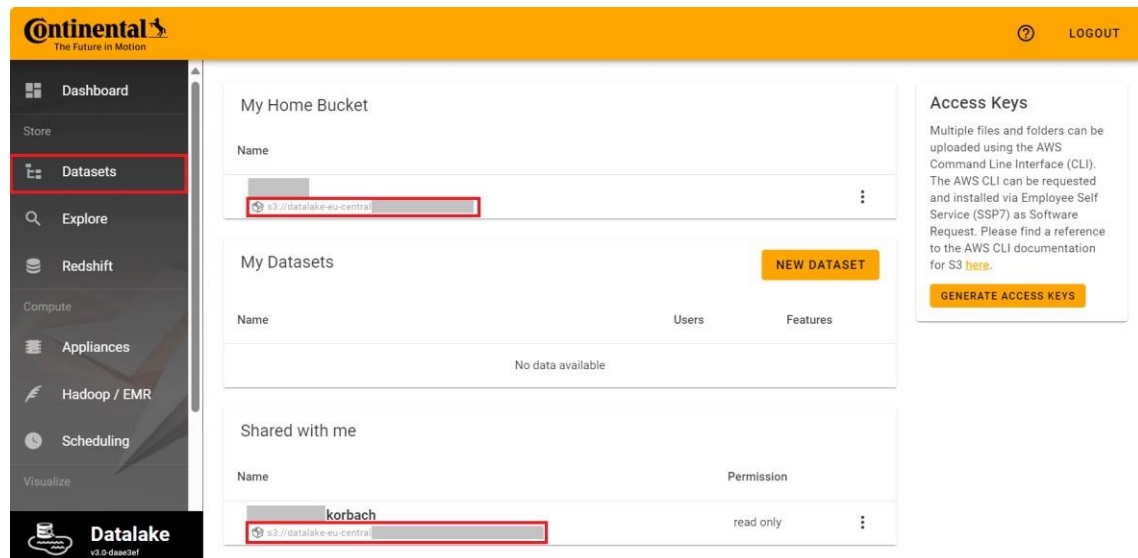


Fig. 4.11 S3 buckets in the Continental Data Lake

In Figure 4.11, you can see the *Datasets section* of Continental's data lake, which contains a bucket at the top of S3 that has been assigned to our account and at the bottom of S3 a bucket that is shared with us. The latter contains Continental process data from the production plant in Korbach, with which we carried out experiments. The upper S3 bucket assigned to the account is empty and for our purposes we were only getting acquainted with the possibilities of work. If necessary, users can create multiple datasets by clicking on the *New dataset button*.

After clicking on a specific S3 bucket, its contents can be seen. For our purposes, we displayed the data stored in a shared bucket. Part of its interior can be observed in Figure 4.12. The data is grouped according to certain criteria into folders. They are primarily divided into raw data (*raw\_data*) and transformed data. Figure 4.12 shows the part of the production data falling within the raw data domain. The aforementioned dataset contains data from a period of three years – from 2020 to 2023.

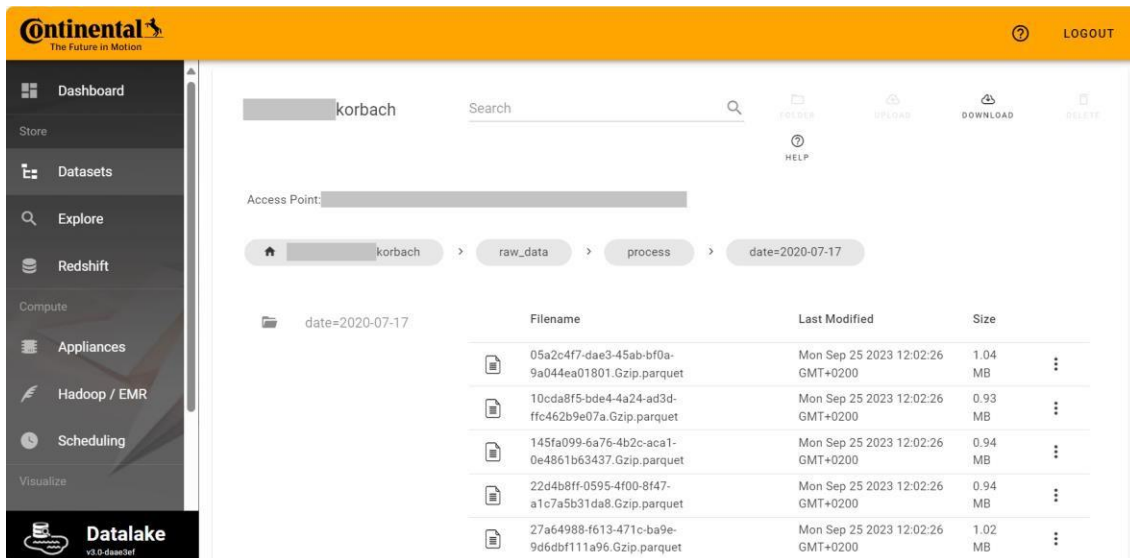


Fig. 4.12 Approximation of the dataset from the production plant in Korbach

Data from a specific bucket or part of it (e.g. *raw\_data*) can also be displayed directly in the SSH console of the EC2 instance created by us. This requires knowledge of basic Linux commands and commands for working with AWS S3 services. The specific command for listing all bucket objects of the Korbach production plant (also shown in Figure 4.13) is as follows [75]:

```
aws s3 ls s3://datalake-eu-centralXXX/XXXkorbach/ --recursive --human-readable --summarize
```

Note: For privacy reasons, the full name of the Korbach data bucket is not provided. Some places are covered by X symbols.

```
[ec2-user@ ~]$ aws s3 ls s3://datalake-eu-centralXXX/XXXkorbach/ --recursive --human-readable --summarize
```

Fig. 4.13 Command to list all items of the selected S3 bucket

As you can see, the basic Linux `ls` command is used here to list items in a given directory, and it is necessary to specify `aws s3` before it (to reserve work with the S3 service). The name of the bucket is indicated after the specified command, starting with `s3://`. In addition, we also used the filters `--recursive` for listing all objects, `--human-readable` for better readability (the size of individual objects is specified), and `--summarize` for listing the total number of objects and the total size of the dataset. A listing of the last S3 bucket items of the Korbach production plant, as well as the aforementioned summary, can be seen in Figure 4.14.

2023-10-12 06:34:33	1.2 GiB	korbach/transformed/cu/CU_2021-02-01_to_2021-03-31.parquet
2023-10-12 06:34:33	1.1 GiB	korbach/transformed/cu/CU_2021-04-01_to_2021-05-31.parquet
2023-10-12 06:35:03	1.2 GiB	korbach/transformed/cu/CU_2021-06-01_to_2021-07-31.parquet
2023-10-10 06:10:12	1.3 GiB	korbach/transformed/cu/CU_2021-08-01_to_2021-09-30.parquet
2023-10-10 06:10:12	1.2 GiB	korbach/transformed/cu/CU_2021-10-01_to_2021-11-30.parquet
2023-10-10 06:10:12	1.2 GiB	korbach/transformed/cu/CU_2021-12-01_to_2022-01-31.parquet
2023-10-10 06:10:12	1.3 GiB	korbach/transformed/cu/CU_2022-02-01_to_2022-03-31.parquet
2023-10-09 07:46:31	1.2 GiB	korbach/transformed/cu/CU_2022-04-01_to_2022-05-31.parquet
2023-10-09 07:46:31	1.3 GiB	korbach/transformed/cu/CU_2022-06-01_to_2022-07-31.parquet
2023-10-09 07:46:31	1.3 GiB	korbach/transformed/cu/CU_2022-08-01_to_2022-09-30.parquet
2023-10-09 07:46:31	1.3 GiB	korbach/transformed/cu/CU_2022-10-01_to_2022-11-30.parquet
2023-10-09 07:46:31	630.1 MiB	korbach/transformed/cu/CU_2022-12-01_to_2022-12-31.parquet
2023-09-26 12:19:30	1.5 GiB	korbach/transformed/cu/CU_2023-01-01_to_2023-03-13.parquet
2023-09-26 12:49:17	520.4 MiB	korbach/transformed/cu/profiling/CU_2023-01-01_to_2023-03-13_full_sample.html
2023-10-06 14:16:40	28.9 MiB	korbach/transformed/cu/profiling/CU_2023-01-01_to_2023-03-13_minimal.html
2023-09-21 07:56:59	1.8 GiB	korbach/transformed/ex/EX_2020-07-17_to_2021-07-16.parquet
2023-10-04 07:29:17	1.9 GiB	korbach/transformed/ex/EX_2021-07-17_to_2022-07-16.parquet
2023-10-06 14:17:37	1.3 GiB	korbach/transformed/ex/EX_2022-07-17_to_2023-03-13.parquet
2023-09-22 09:49:13	65.6 MiB	korbach/transformed/ex/profiling/EX_2020-07-17_to_2021-07-16.html
2023-09-29 07:39:49	1.0 GiB	korbach/transformed/tb/TB2_2020-07-17_to_2021-07-16.parquet
2023-09-29 07:40:12	782.0 MiB	korbach/transformed/tb/TB2_2021-07-17_to_2022-07-16.parquet
2023-09-27 09:12:09	1.7 GiB	korbach/transformed/tb/km.parquet
2023-09-27 09:11:27	608.4 MiB	korbach/transformed/tb/pu_2022-07-17_to_2023-03-13.parquet
Total Objects: 46425		
Total Size: 77.6 GiB		

Fig. 4.14 Listing of individual objects S3 buckets of the production plant in Korbach together with summarization

Our dataset contains a total of 46,425 objects, the size of which is equal to 77.6 GB.

### 4.2.3 Cluster

To create a *cluster*, we have to go to the Hadoop / EMR section of the Continental data lake. Similar to instances, there is a list of items owned by us. Since we have not created any cluster yet, we have an empty list so far (see figure 4.15). A *cluster* can be created by clicking the *New Hadoop cluster button*.

After clicking on the mentioned button, a window with cluster configuration options will appear (see figure 4.16). First, we entered the name of *the cluster*, again with a lowercase initial letter. We kept the latest version and then we had the opportunity to choose the tools we wanted to be installed within our *cluster*. We chose *Spark, Hadoop, and Tensorflow*. Finally, it is necessary to choose the number of instances that the *cluster will* consist of. There are options 3, 4 or 5 to choose from. For our purposes of taking full advantage of the performance of work with *the Apache Spark cluster*, we have chosen 5 instances. Each of the instances is of the *M5a.xlarge type*, i.e. it consists of 4 processor cores and 16 GB of RAM. After creating a *cluster*, it will be displayed in the list (see figure 4.17).

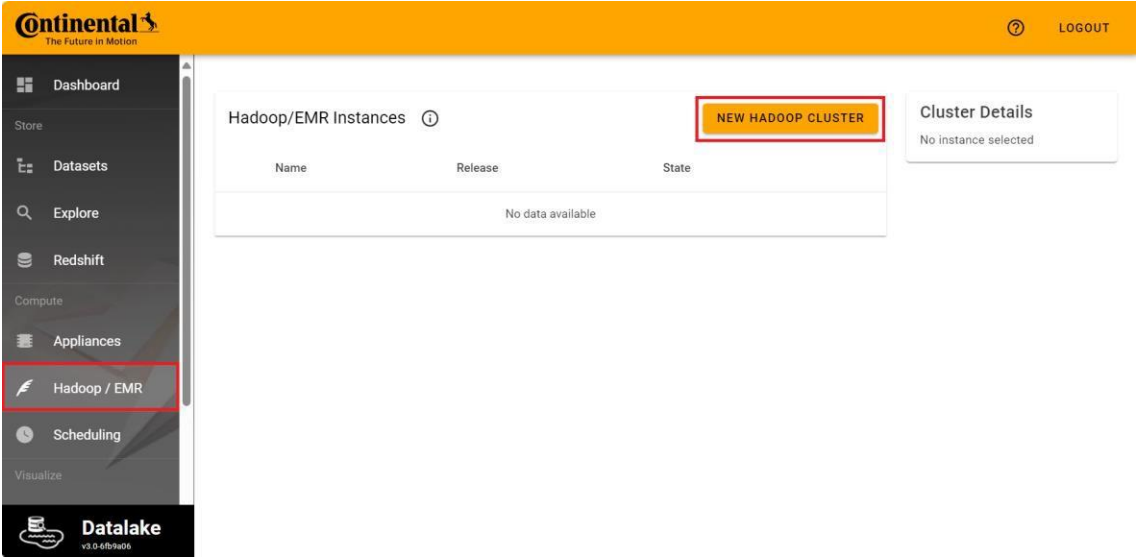


Fig. 4.15 Preview of the *cluster* in the Continental data lake

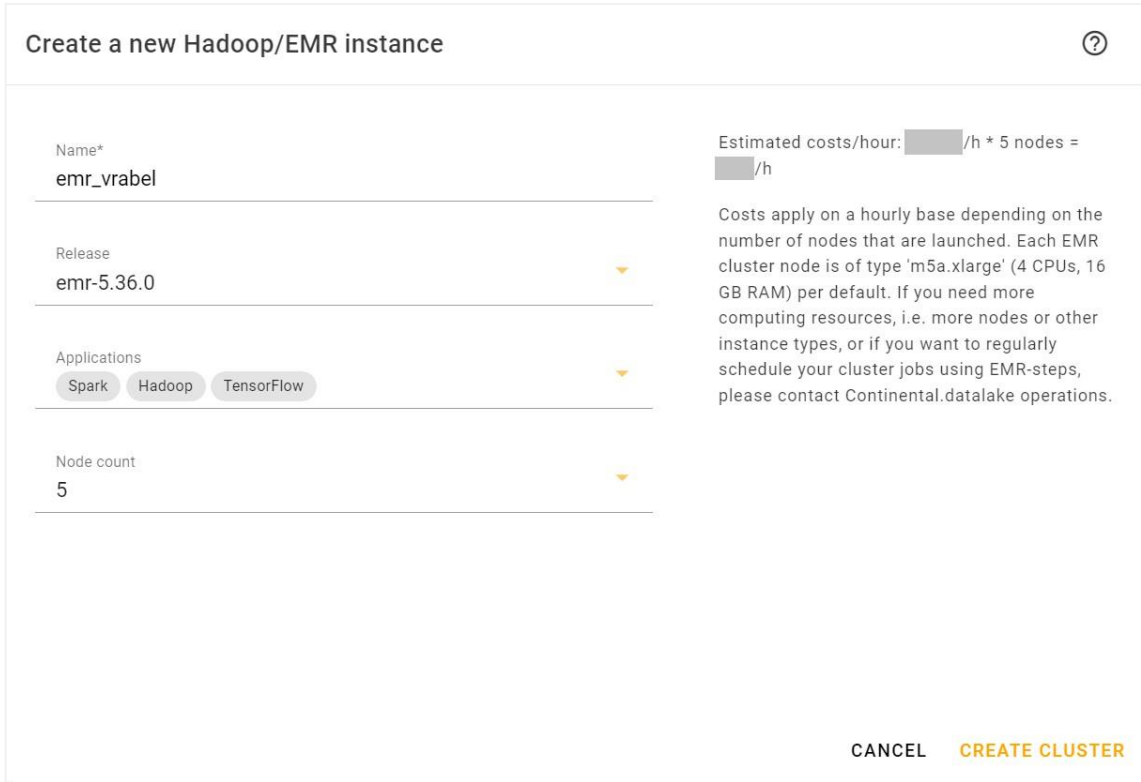


Fig. 4.16 Creating an EMR *cluster*



Name	Release	State
emr_vrabel	emr-5.36.0	WAITING

Fig. 4.17 List of created *clusters*

Unlike EC2 instances, *clusters* cannot be in a shutdown state. Once created, *the cluster* is constantly turned on. Therefore, if we no longer need to use its services, it is important to terminate the existence of *the cluster* in its capabilities (*Terminate cluster*). As a result, we will not be charged.

### 4.3 Data processing of the manufacturing company

In our diploma thesis, we focused on the analysis of transformed data from our S3 data repository containing process data from the production plant in Korbach. The relevant group includes data from three production areas: *curing* with a total of 14 stations, *extrusion* with one station and *tire building* with a total of 4 stations. Each of the production departments contains dozens of parameters the same for each station with millions of values. For this reason, for the purposes of the diploma thesis, we focused only on selected stations from each of the mentioned production areas, while we tried to choose ten parameters in each industry. The stated goal was set by the consultant, so we state in the chapter 5 Results of data analysis summary of all parameters processed by us. At work, we did not have knowledge of the meaning of individual parameters. Our purpose was only to process the given parameters and analyze possible dependencies.

For data analysis, we used to work with the created *cluster*. We secured the connection to it based on the knowledge of its DNS (*Domain Name System*) address. This can be seen by clicking on our *cluster emr\_vrabel* in the list offered by Continental's data lake (see figure 4.18).

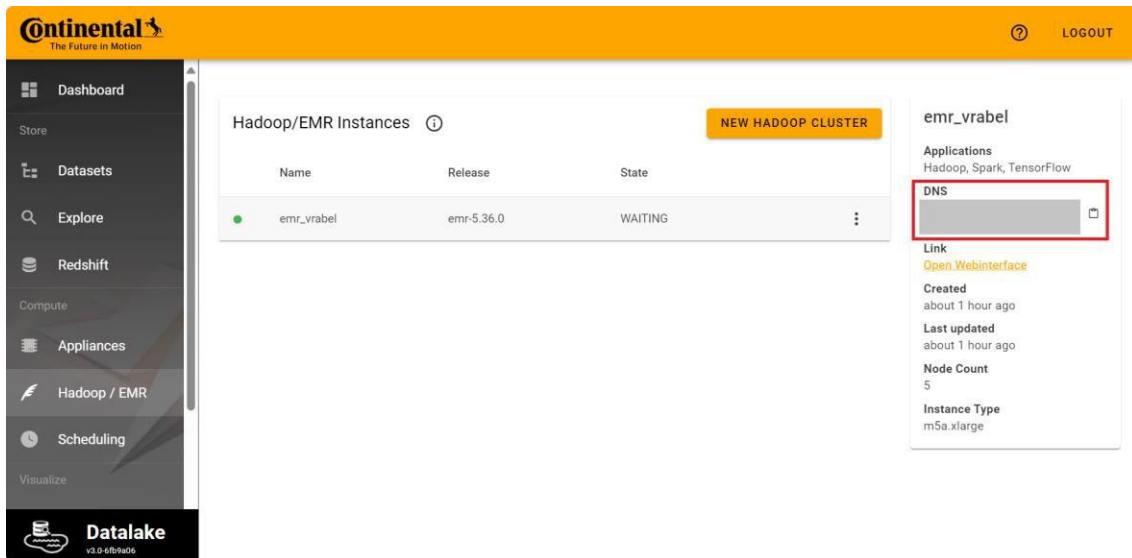


Fig. 4.18 Information about the *emr\_vrabel* cluster

To connect to the *cluster* through its DNS addresses, we used the MobaXterm Windows software. It is a set of tools that allow you to use remote connection. It supports, for example, SSH, X11, RDP (*Remote Desktop Protocol*), VNC (*Virtual Network Computing*), FTP (*File Transfer Protocol*), as well as Unix commands such as `bash`, `ls`, `cat`, `sed`, `grep`, `awk`, `rsync` [76].

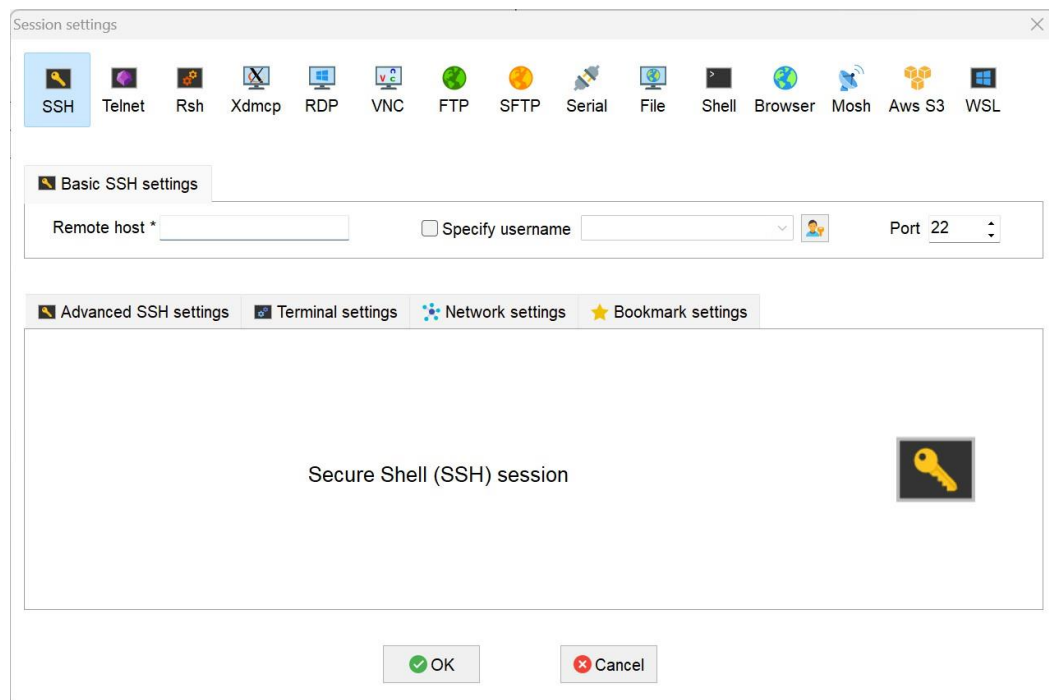


Fig. 4.19 Connecting to a *cluster* via SSH protocol in MobaXterm software

In the MobaXterm software, we created a connection with *the cluster* via the SSH protocol. We filled in the remote *host field* with the DNS address of *the cluster* and also the user field (*username*), where we entered the name of our AWS account.

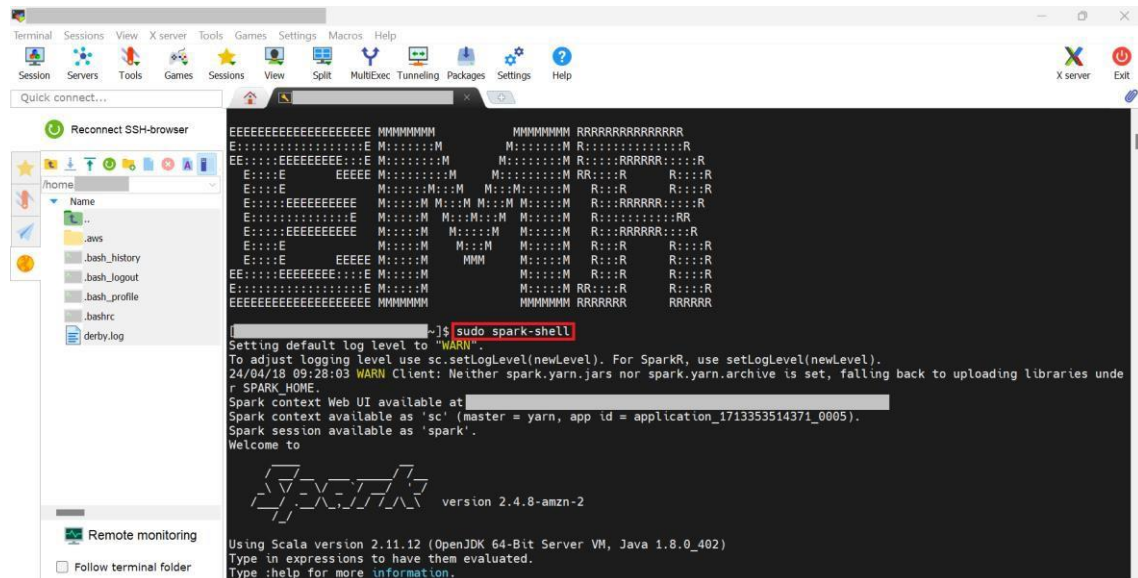


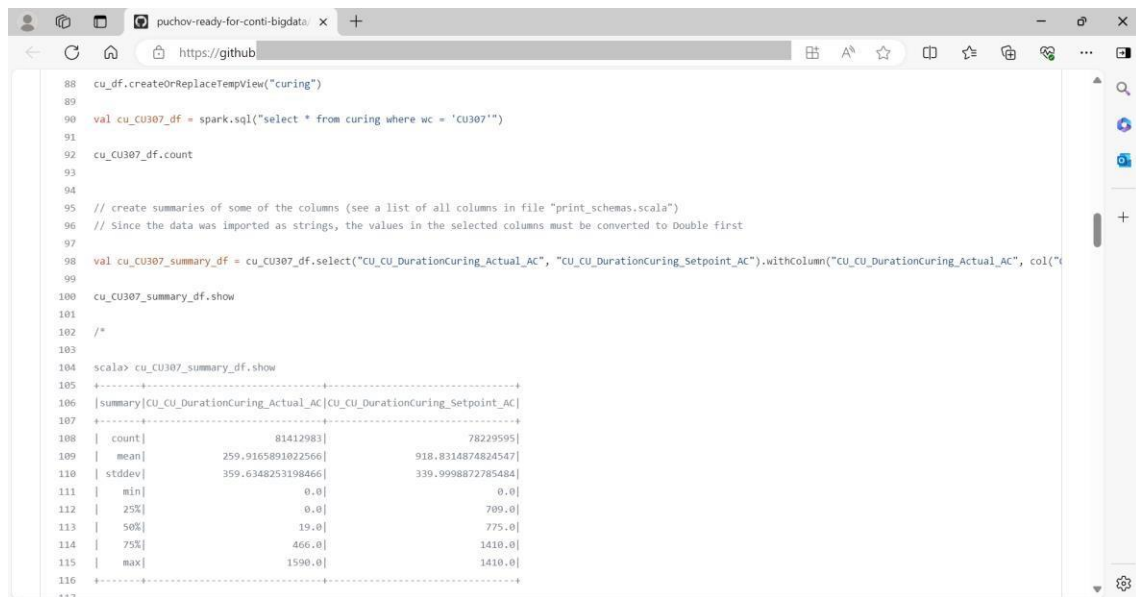
Fig. 4.20 Running *Apache Spark* in an EMR *cluster*

After establishing the connection, we were presented with a console window with welcome EMR symbols. Subsequently, it was necessary to run *Apache Spark* in *the cluster*. The implementation takes place by entering *the bash* command `sudo spark-shell` (see figure 4.20). As can be seen, *Apache Spark* is primarily run in direct interaction with the *Scala programming language*.

The thesis consultant provided us with programs written in the mentioned programming language in the GitHub repository. In the practical part, we used two of the mentioned programs containing the required functions for loading and processing data. Their names can be seen in Table 4.1. The part of the code with the number 2, used to read and summarize the data, can be seen in Figure 4.21.

Table 4.1 Names of programs used by the consultant

number	Name of the programme
1	print_schemas.scala
2	write_data_summaries.scala

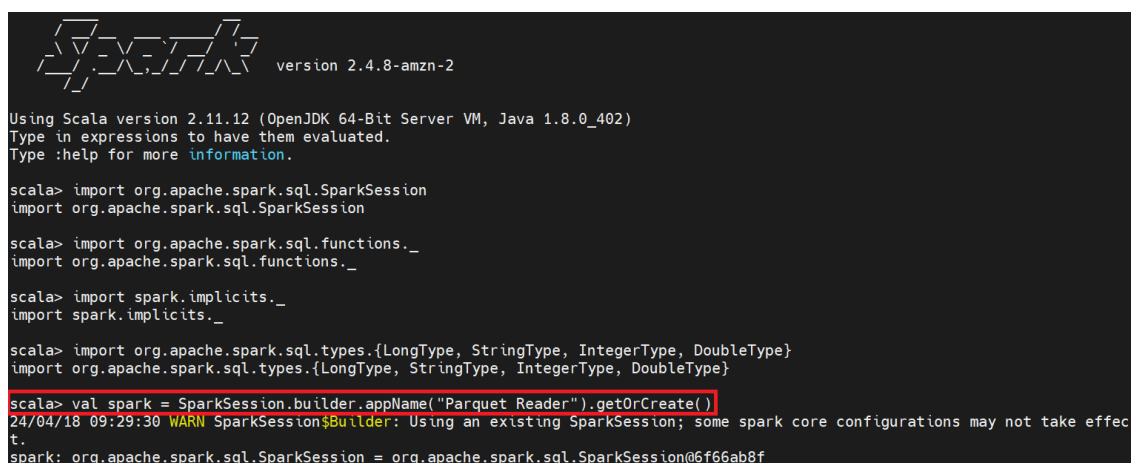


```

88 cu_df.createOrReplaceTempView("curing")
89
90 val cu_CU307_df = spark.sql("select * from curing where wc = 'CU307'")
91
92 cu_CU307_df.count
93
94
95 // create summaries of some of the columns (see a list of all columns in file "print_schemas.scala")
96 // Since the data was imported as strings, the values in the selected columns must be converted to Double first
97
98 val cu_CU307_summary_df = cu_CU307_df.select("CU_CU_DurationCuring_Actual_AC", "CU_CU_DurationCuring_Setpoint_AC").withColumn("CU_CU_DurationCuring_Actual_AC", col("
99
100 cu_CU307_summary_df.show
101
102 /*
103
104 scala> cu_CU307_summary_df.show
105
106 |summary|CU_CU_DurationCuring_Actual_AC|CU_CU_DurationCuring_Setpoint_AC|
107 |-----|-----|-----|
108 | count|                81412983|                78229595|
109 |  mean|    259.9165891022566|    918.8314874824547|
110 | stddev|    359.6348253198466|    339.9998872785484|
111 |  min|                0.0|                0.0|
112 | 25%|                0.0|                789.0|
113 | 50%|                19.0|                775.0|
114 | 75%|                466.0|                1410.0|
115 |  max|                1590.0|                1410.0|
116 |-----|-----|-----|
117

```

Fig. 4.21 The *GitHub* section of the `write_data_summaries.scala` code



```

Databricks version 2.4.8-amzn-2

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_402)
Type in expressions to have them evaluated.
Type :help for more information.

scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> import org.apache.spark.sql.functions._
import org.apache.spark.sql.functions._

scala> import spark.implicits._
import spark.implicits._

scala> import org.apache.spark.sql.types.{LongType, StringType, IntegerType, DoubleType}
import org.apache.spark.sql.types.{LongType, StringType, IntegerType, DoubleType}

scala> val spark = SparkSession.builder.appName("Parquet Reader").getOrCreate()
24/04/18 09:29:30 WARN SparkSession$Builder: Using an existing SparkSession; some spark core configurations may not take effect.
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@6f66ab8f

```

Fig. 4.22 Importing libraries and creating a connection

Figure 4.22 shows the import of libraries needed to use the desired features in *Apache Spark*. The marked command is required to create a session with *Apache Spark*.

In order to be able to process the data, we needed to have the names of the relevant parameters for individual production sectors listed. Based on this, we knew how many parameters the mentioned areas were described by and by what designation to approach them. For the extraction, we used program number 1 from Table 4.1. Part of it can be seen in Figure 4.23, where we had the collected data retrieved before the actual statement from the press shop from the period 17.7.2020 to 30.9.2020. In total, the area is described by 58 parameters (including timestamp, station designation, and index).



In the following parts necessary for the listing of stations together with the number of their values, for creating a subset of data with only values from one specific station and for processing selected parameters together with the list of results, we used program number 2 from Table 4.1.

Figure 4.24 shows the command to retrieve all data from the production area of the press shop over a period of three years. We will also show the other functions used on the given data. In Figure 4.25, we have listed all stations from the mentioned branch of production along with the number of values for each of its parameters. As can be seen, a total of 15 lines of stations were listed, but one of them is probably incorrect data – line 06708 with 3 values. According to the correctness, the stations from the press shop area are marked with the initial symbols CU and show values in the number of over 80 million. For our purposes, we chose CU306, CU307 and CU310 stations for data processing.

```
scala> cu_df.createOrReplaceTempView("curing")
scala>
scala> val cu_CU306_df = spark.sql("select * from curing where wc = 'CU306'")
cu_CU306_df: org.apache.spark.sql.DataFrame = [timestamp: timestamp, wc: string ... 57 more fields]
scala> cu_CU306_df.count
res2: Long = 81441310
```

Fig. 4.26 Total number of values for individual parameters of the CU306 station

We created a variable `cu_CU306_df`, into which we selected data from only one of the stations using an SQL statement. In our case, we chose the station CU306. For checking, we had the number of values in its parameters listed (see Figure 4.26) and the same number was printed as in the case of the number of values for the list of all stations shown in Figure 4.25.

```
scala> val cu_CU306_summary_df = cu_CU306_df.select("CU_CU_DurationVacuum_Actual_AC", "CU_MO_DurationShaping_Actual_AC").withColumn("CU_CU_DurationVacuum_Actual_AC", col("CU_CU_DurationVacuum_Actual_AC").cast(DoubleType)).withColumn("CU_MO_DurationShaping_Actual_AC", col("CU_MO_DurationShaping_Actual_AC").cast(DoubleType)).summary()
cu_CU306_summary_df: org.apache.spark.sql.DataFrame = [summary: string, CU_CU_DurationVacuum_Actual_AC: string ... 1 more field]
scala> cu_CU306_summary_df.show
```

Fig. 4.27 Modification of the selected two parameters of the CU306 station together with the command to extract the summarization

Subsequently, we carried out a summarization. It is a `summary()` function provided by the *Scala programming language*. We `cu_306_summary_df` into the new variable selected data for two selected parameters. Since the data set is stored in the

*string* format, we needed to convert it to the double data type and then we could apply the appropriate function (see figure 4.27).

After listing the summarization, it showed us a table with statistical data of selected parameters (see figure 4.28). This is the total number of counts included in the calculations. It is worth noting that the given number does not match the total number of measurement values for a given station. This is because some data contains a `null` value, and the `summary()` function does not take such into account. Subsequently, the mean (in English *mean*), standard deviation (*standard deviation in English*), the minimum together with the maximum value and finally the values of three quartiles are displayed. In general, the quartile expresses the maximum that can range from the minimum to the quartile in question. For example, the first quartile (25%) for our case in Figure 4.28 expresses that the first 25% of the values from the minimum reach a level from 0 (the value of the minimum) to 5 (the value of the first quartile). The second quartile (50%) referred to as the median works in a similar way, as well as the third quartile (75%).

summary	CU_CU_DurationVacuum_Actual_AC	CU_MO_DurationShaping_Actual_AC
count	78251243	81434110
mean	16.55645067516691	10.919932912633293
stddev	82.1150599461577	4.84428878092647
min	0.0	0.0
25%	5.0	8.0
50%	5.0	11.0
75%	5.0	13.0
max	600.0	178.0

Fig. 4.28 Extract of summarization of selected two parameters of the CU306 station

We applied the summarization function to all three mentioned stations of the press shop, and we used it on the same 10 parameters for the purpose of comparing the results.

We used the same procedure that we have outlined here for the rest of the production processes. For the extrusion process, we had only one EX071 station available (see Figure 4.29). In total, the production area is described by 229 parameters (including time stamp, station designation and index). We had to go through several of them, because they mostly contained values of 0, which are not interesting from the point of view of analysis.



```

+-----+-----+
|   wc |   count |
+-----+-----+
| EX071 | 80258076 |
+-----+-----+

```

Fig. 4.29 List of stations with the number of values in the extrusion production area

We proceeded in a similar way with the production area of ready-to-wear. We chose the TBPH1 and TBPH2 stations for the analysis. In total, the industry is described by 51 parameters (including timestamp, station designation and index), but we were able to use only 6 of them. The remaining ones (apart from the above in parentheses) contained values of 0 or null (see figure 4.31), i.e. they were either uninteresting from the point of view of analysis or could not be used.

```

+-----+-----+
|   wc |   count |
+-----+-----+
| TBKH1 | 81841405 |
| TBPH2 | 74314369 |
| TBKH2 | 81957232 |
| TBPH1 | 81271609 |
+-----+-----+

```

Fig. 4.30 List of stations with the number of values in the ready-to-wear production area

```

+-----+-----+-----+
| summary | TB2_TR_AreaTreadEdgeLeft_Actual_AC | TB2_TR_LengthOnServicer_Actual_AC |
+-----+-----+-----+
| count | 0 | 0 |
| mean | null | null |
| stddev | null | null |
| min | null | null |
| 25% | null | null |
| 50% | null | null |
| 75% | null | null |
| max | null | null |
+-----+-----+-----+

```

Fig. 4.31 Parameters of the TBPH2 station with null values



## 5 RESULTS OF DATA ANALYSIS

In this chapter, we will look at the results obtained after applying the `summary()` function to selected parameters of selected stations in three production sectors. We will also show additional information resulting from the data obtained and also compare the data of specific parameters between individual stations.

As a reminder, we obtained the results using the functions contained in the programs from the thesis consultant, the names of which are listed in Table 4.1. To find out the number of parameters of individual production areas and their listing, from which we chose the parameters for analysis, we used program number 1. To select data from a specific station, to determine the number of values for its parameters and for their subsequent processing, we used program number 2.

Note 1: We also used *box plots* for the analysis. We did not mention *outliers* in them. In the results, only the minimum and maximum (if any) can be read from the tables.

Note 2: As the output of the analysis will be further used by Continental, we have left the English markings in the tables.

### 5.1 Vulcanization or Curing

For the stamping process, we focused on CU306, CU307 and CU310 stations. Let's take a look at the conclusion of the analysis for the ten selected parameters:

#### a) CU\_CU\_DurationVacuum\_Actual\_AC

Table 5.1 Analysis of stations CU306, CU307, CU310 for parameter a)

summary	CU306	CU307	CU310
Count	78,251,243	78,229,595	78,629,111
mean	16.6	12.4	11.9
stddev	82.1	65.8	63.8
Min	0	0	0
25%	5	5	5
50%	5	5	5
75%	5	5	5
max	600	600	600
nulls	3,190,067	3,192,388	2,783,598
skewness	right	right	right

As can be seen, compared to the table with the values obtained by the summary() function shown in Figure 4.28, we added two rows to the final evaluation – a row showing the number of null values occurring between the values of the parameter for a particular station, and also a row describing the type of skewness for a given data distribution). We were able to determine the skewness from the summarization by comparing the median and mean values. In most cases, if the value of the average is greater than the value of the median, it is a right-sided or positive skew. It is also usually true that if the mean is less than the median, it is a left-sided or negative skew [77].

From the values of the minimum, individual quartiles and maximum shown in Table 5.1, it can be concluded that parameter a) mostly acquires and probably should acquire a value of 5. It contains some outliers that make it have a true skew. Compared to the rest of the monitored stations, CU306 has higher values of diameter and standard deviation, on the basis of which it can be concluded that it either has a larger number of outliers towards the maximum or acquires them at a higher level.

#### **b) CU\_MO\_DurationShaping\_Actual\_AC**

Table 5.2 Analysis of stations CU306, CU307, CU310 for parameter b)

summary	CU306	CU307	CU310
Count	81,434,110	81,414,783	81,394,738
mean	10.9	13.0	9.4
stddev	4.8	8.4	4.6
Min	0	0	0
25%	8	10	8
50%	11	12	9
75%	13	16	10
max	178	1,108	362
nulls	7,200	7,200	17,971
skewness	left	right	right

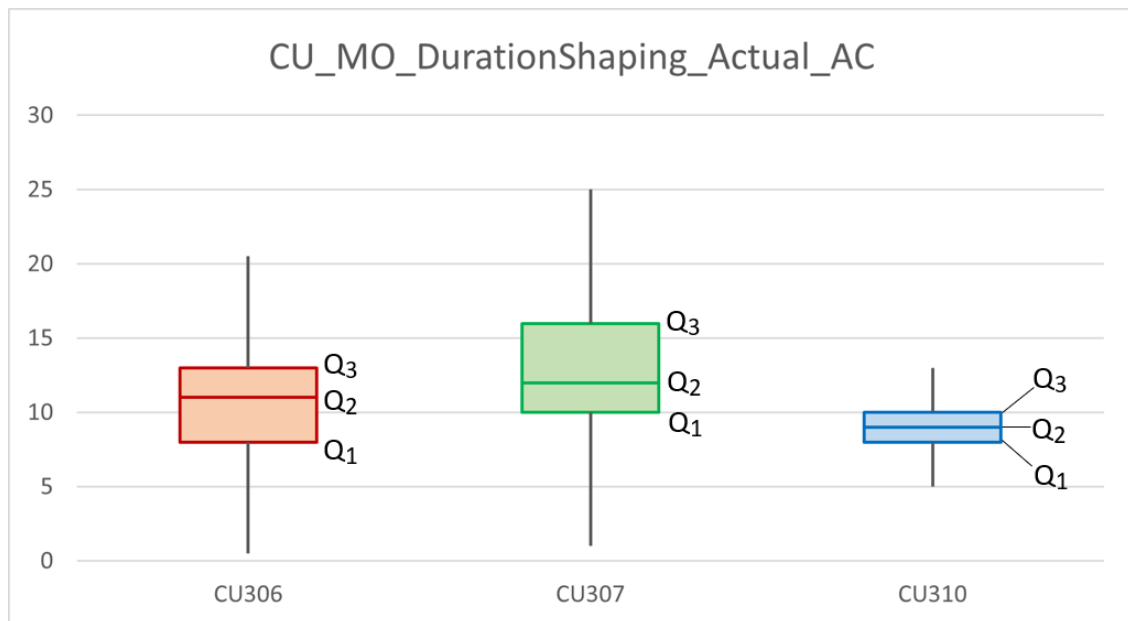


Fig. 5.1 Box diagram of stations CU306, CU307, CU310 for parameter b)

For parameter b), lower standard deviation numbers can be noted, thanks to which its values are more distributed around the mean. The CU306 station acquires approximately the same values of diameter and median, thanks to which we could consider the given data distribution to be normal with a slight skew to the left. Similarly, we can consider the CU310 station as a station with normally distributed data with a slight skew to the right. In the case of CU307 stations, the difference between the diameter and the median is already greater. It acquires a true obliqueness. The phenomenon can be observed in Table 5.2 as well as in the graphical representation in Figure 5.1, where approximately the same magnitude of the distances between the Q1-Q2 and Q2-Q3 quartiles for the CU306 station and the same distances between the Q1-Q2 and Q2-Q3 quartiles for the CU310 station can be seen. In the case of the CU307 station, the asymmetry of copper can be seen with the stated distances. The Q2-Q3 distance is higher, from which the oblique to the right can be read.

### c) CU\_CU\_PercentageControlValveOpenB\_Actual\_AC

Table 5.3 Analysis of stations CU306, CU307, CU310 for parameter c)

summary	CU306	CU307	CU310
Count	81,435,910	81,416,583	81,407,309
mean	18.0	14.4	15.8
stddev	23.6	21.9	26.0
Min	0	0	0
25%	6	4	3
50%	12	8	7
75%	20	14	15
max	100	100	100
nulls	5,400	5,400	5,400
skewness	right	right	right

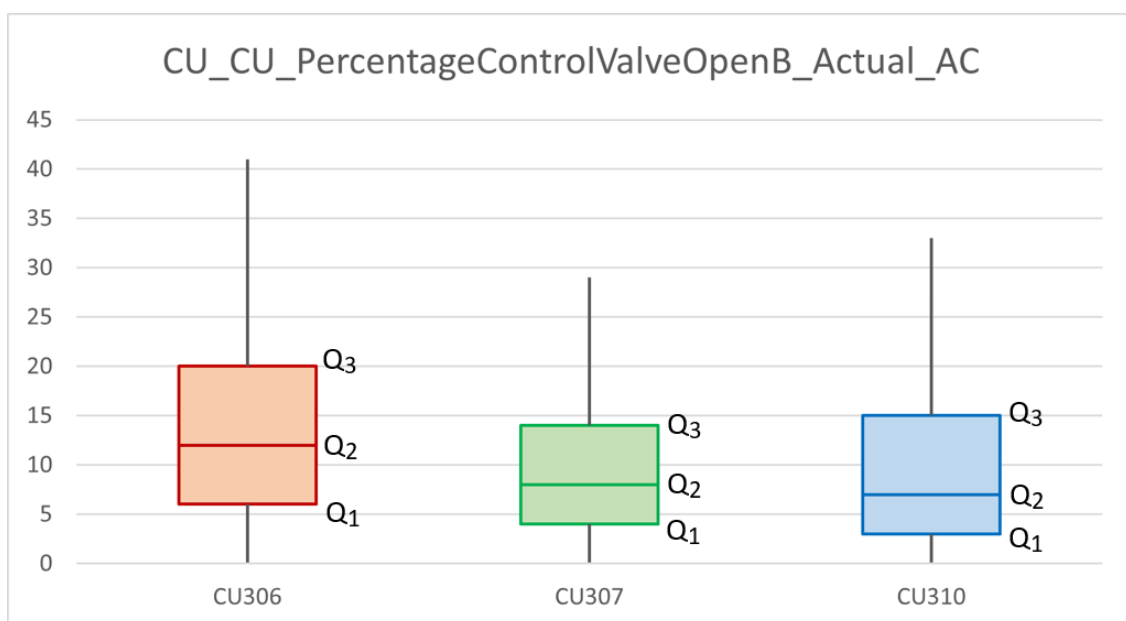


Fig. 5.2 Box diagram of stations CU306, CU307, CU310 for parameter c)

In the case of parameter c), it is easy to verify the correctness of the acquired data. It is a parameter expressing percentages, so its values must fall within the range from 0 to 100. This condition is met.

### d) CU\_CU\_PositionClosingStop1\_Actual\_AC

Table 5.4 Analysis of stations CU306, CU307, CU310 for parameter d)

summary	CU306	CU307	CU310
Count	81,437,710	81,418,383	81,409,109
mean	902.4	847.3	919.7
stddev	626.2	609.6	652.8
Min	0	0	0
25%	501	500	430
50%	503	500	502
75%	1,880	1,854	1,876
max	1,998	1,907	1,989
nulls	3,600	3,600	3,600
skewness	right	right	right

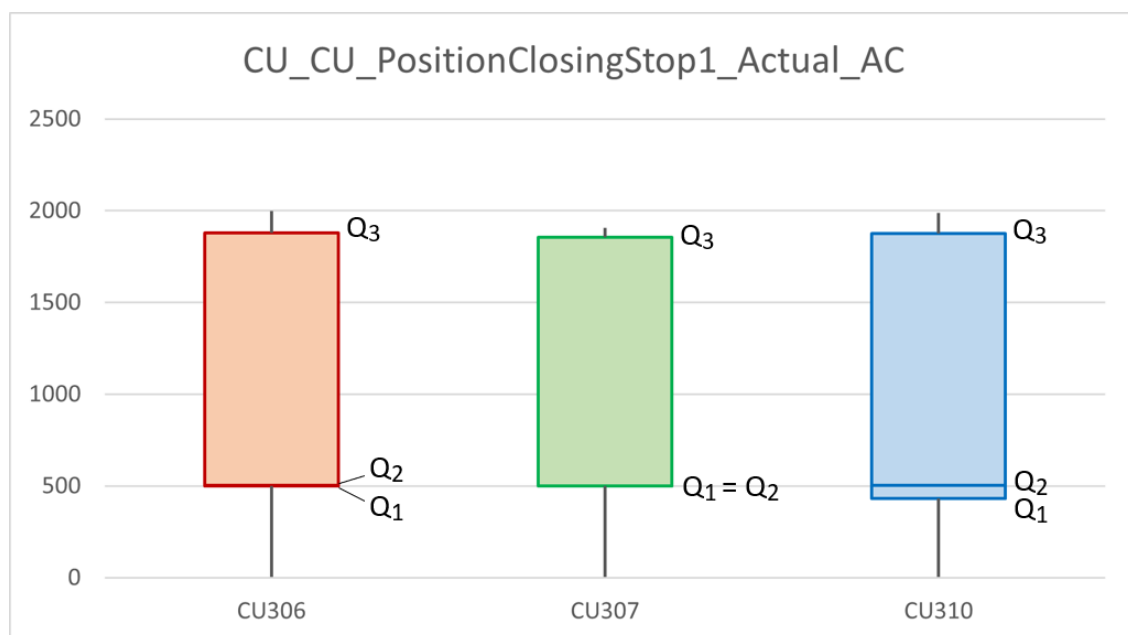


Fig. 5.3 Box diagram of stations CU306, CU307, CU310 for parameter d)

When we look at the table values of the stations for parameter d) and also at their graphical representation, we can see a similarity between them. They differ only slightly. It is worth noting almost the same value of the first and second quartile for the CU306 station and the same value of 500 for the mentioned quartiles of the CU307 station. It is a parameter expressing the position and it can therefore be concluded that it acquires the median value for the CU306 station at level 503, for the station CU307 at level 500 and for the station CU310 at level 502.

### e) CU\_CU\_TemperatureADTT\_Actual\_AC

Table 5.5 Analysis of stations CU306, CU307, CU310 for parameter e)

summary	CU306	CU307	CU310
Count	81,437,710	81,418,383	81,398,338
mean	0.072	0.065	0.048
stddev	0.295	0.359	0.303
Min	-3.14	-6.42	-21.28
25%	0	0	0
50%	0	0	0
75%	0.07	0.07	0.03
max	5.25	5.90	5.90
nulls	3,600	3,600	14,371
skewness	right	right	right

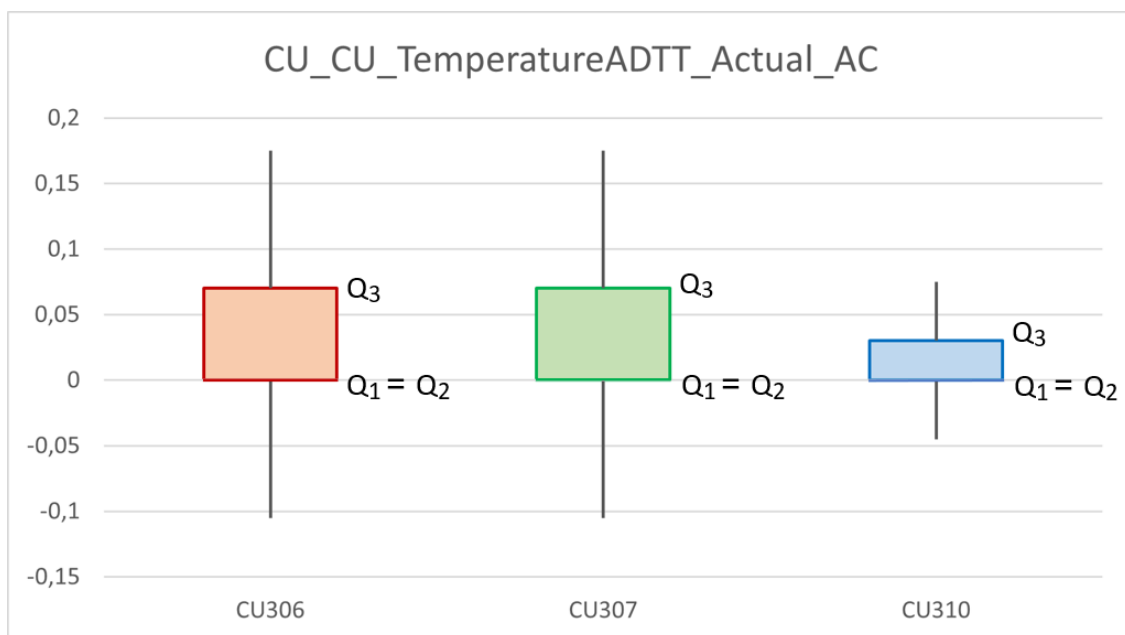


Fig. 5.4 Box diagram of stations CU306, CU307, CU310 for parameter e)

Parameter (e) represents the temperature in the production process. For this reason, negative minimum values for monitored stations are suspicious. This is probably a measurement error. When we notice the values of quartiles and maximums, they are quite similar. Based on the values of the quartiles, it can be stated that the most frequently measured value is 0, or its very close surroundings towards positive values. All stations have data distributed with a right skew.

### f) CU\_CU\_PressureInBladder\_Actual\_AC

Table 5.6 Analysis of stations CU306, CU307, CU310 for parameter f)

summary	CU306	CU307	CU310
Count	81,435,910	81,416,583	81,407,309
mean	7.2	7.9	6.1
stddev	7.8	8.4	7.3
Min	-1	-1	-1.05
25%	0.02	0	-0.04
50%	0.42	1.26	0.03
75%	14.77	15.74	13.84
max	22.62	23.06	22.78
nulls	5,400	5,400	5,400
skewness	right	right	right

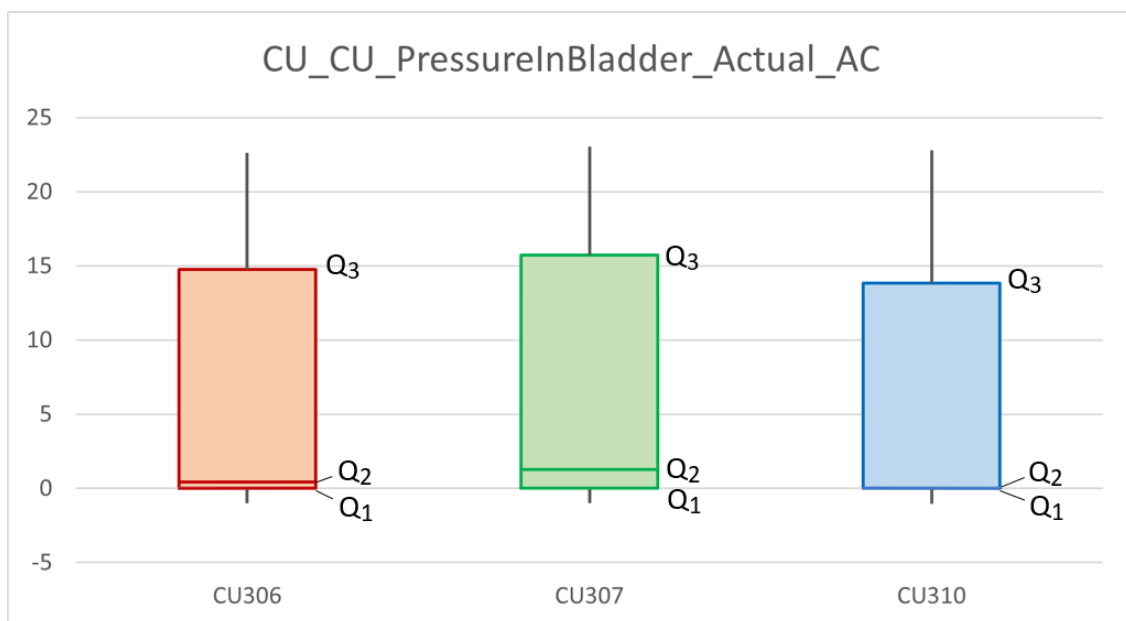


Fig. 5.5 Box diagram of stations CU306, CU307, CU310 for parameter f)

Parameter f) represents the pressure in the production process, and as in the case of temperature, the negative values of the minima and the first quartile at the CU310 station are suspicious. Most likely, these are mistakes.

### g) CU\_CU\_StatusValveHighPressureSteam\_Actual\_AC

Table 5.7 Analysis of stations CU306, CU307, CU310 for parameter g)

summary	CU306	CU307	CU310
Count	81,437,710	81,418,383	81,409,109
mean	14.4	27.4	23.1
stddev	24.7	20.8	25.1
Min	0	0	0
25%	5	20	14
50%	8	26	17
75%	11	29	20
max	100	100	100
nulls	3,600	3,600	3,600
skewness	right	right	right

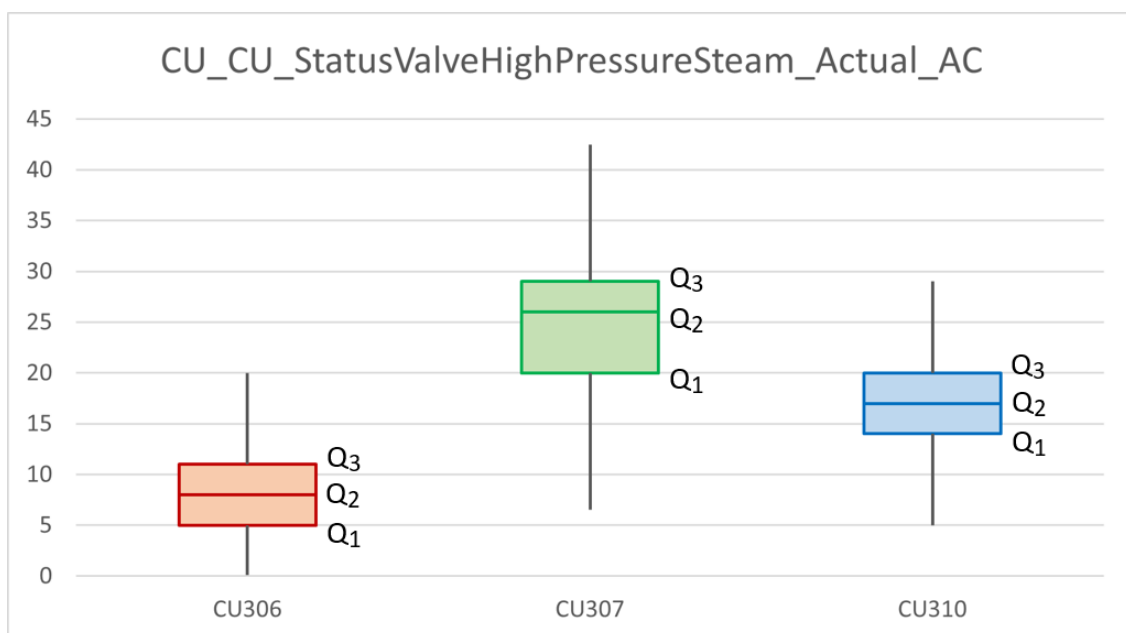


Fig. 5.6 Box diagram of stations CU306, CU307, CU310 for parameter g)

For parameter g), the difference in the acquired values between the respective stations can be seen, especially on the basis of a graphical representation. All of the above stations acquire levels in the range from 0 to 100, but differ in the values of quartiles, and thus in the mean and standard deviation. The CU306 station has the most values concentrated in the lower levels. Up to 75% of the minimum lies in the range from 0 to 11. Station CU307 has the interval up to the third quartile shifted highest of the monitored stations,



to a value of 29. The CU310 station is located between the other two stations in its values. Its third quartile lies at the level of 20.

#### **h) CU\_DM\_DurationDemolding\_Actual\_AC**

Table 5.8 Analysis of stations CU306, CU307, CU310 for parameter h)

<b>summary</b>	<b>CU306</b>	<b>CU307</b>	<b>CU310</b>
Count	78,256,643	78,234,995	78,634,511
mean	2.9	2.9	3.0
stddev	0.4	0.4	0.4
Min	0	0	0
25%	3	3	3
50%	3	3	3
75%	3	3	3
max	3	3	3
nulls	3,184,667	3,186,988	2,778,198
skewness	left	left	left

The monitored stations are almost identical for the parameter h). The values of quartiles and maximums show that for all stations the relevant parameter should have a value of 3.

**i) CU\_LOA\_DegreeInclinationSensorLoader\_Actual\_AC**

Table 5.9 Analysis of stations CU306, CU307, CU310 for parameter i)

summary	CU306	CU307	CU310
Count	81,434,110	81,414,783	81,405,509
mean	0.027	0.057	0.027
stddev	0.266	0.176	0.189
Min	-12.82	-20.03	-8.12
25%	0.01	0.02	-0.04
50%	0.03	0.06	0
75%	0.05	0.08	0.05
max	13.15	37.12	11.24
nulls	7,200	7,200	7,200
skewness	left	left	right

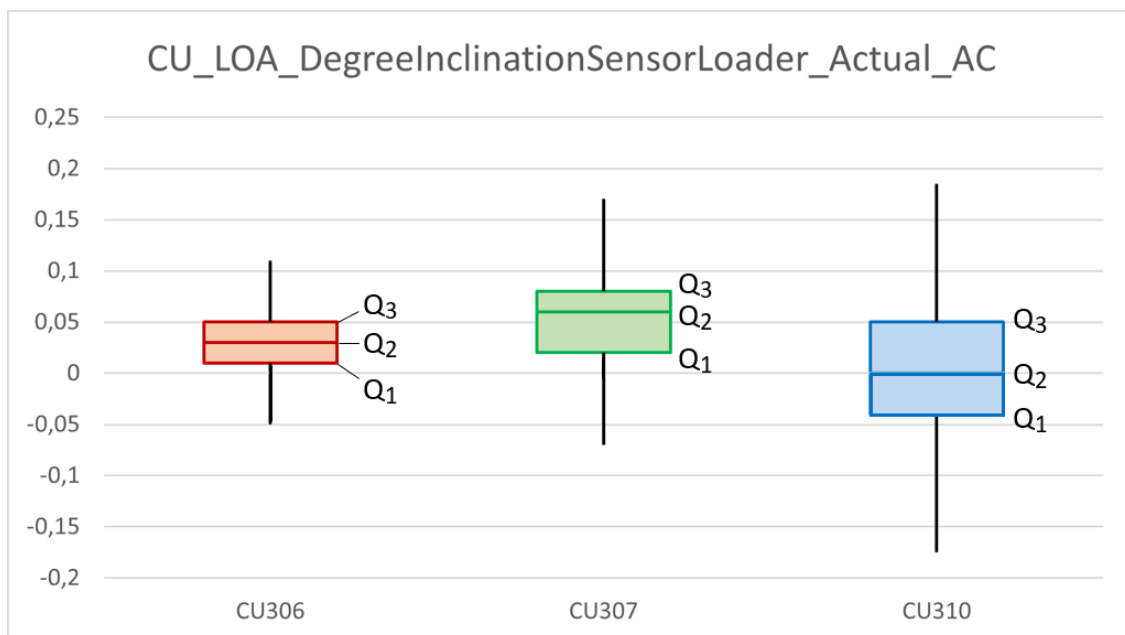


Fig. 5.7 Box diagram of stations CU306, CU307, CU310 for parameter i)

Based on the results of parameter (i), it can be concluded that the stations have most values close to zero. The quartile levels correspond to this. When we compare the sizes of the median and the mean, we can conclude that in the case of the CU306 and CU307 stations, it is a normal distribution of data with a slight skewness to the left. With the CU310 station, the difference between the mean and the median is slightly higher, and the data distribution is skewed to the right.

### j) CU\_MO\_ForceClosing\_Actual\_AC

Table 5.10 Analysis of stations CU306, CU307, CU310 for parameter j)

summary	CU306	CU307	CU310
Count	81,434,110	81,414,783	81,405,509
mean	752.7	739.2	699.4
stddev	413.5	459.1	373.6
Min	0	0	0
25%	362.2	304.4	373.2
50%	530.6	941.9	522.6
75%	1,176.9	1,136.7	1,086.8
max	1,493,0	1,803,4	1,607.3
nulls	7,200	7,200	7,200
skewness	right	left	right

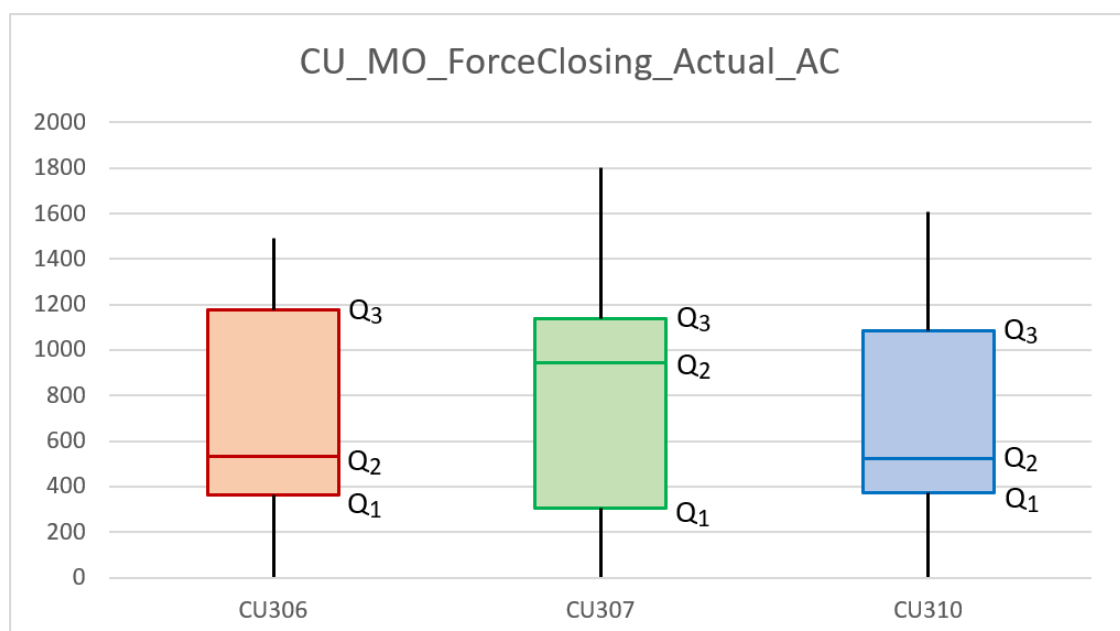


Fig. 5.8 Box diagram of stations CU306, CU307, CU310 for parameter j)

In the graphical representation of parameter j), the difference between the respective stations can be seen at first glance. Specifically, it is the location of the median. While the CU306 and CU310 stations have it closer to the first quartile, at levels of 530.6 and 522.55 respectively; station CU307 has a median placed closer to the third quartile, at 941.92. Based on Figure 5.8, we can see that for stations CU306 and CU310, the distance between the Q1-Q2 quartiles is less than the distance between the Q2-Q3 quartiles. With the CU307 station, it is the other way around. Of which

The skew direction for the respective distributions is also indicated. Stations CU306 and CU310 have a skew to the left, while station CU307 has a skew to the right.

## 5.2 Extrusion

In the extrusion process, we had only one EX071 station available. We will show and compare the final values for ten parameters belonging to the given production area:

### a) EX\_CWB\_PositionSidewall1WindingDancer\_Actual\_AC

Table 5.11 Analysis of the EX071 station for parameter a)

summary	EX071
Count	80,258,076
mean	-81.0
stddev	36.8
Min	-100
25%	-100.0
50%	-100.0
75%	-78.4
max	100
nulls	0
skewness	right

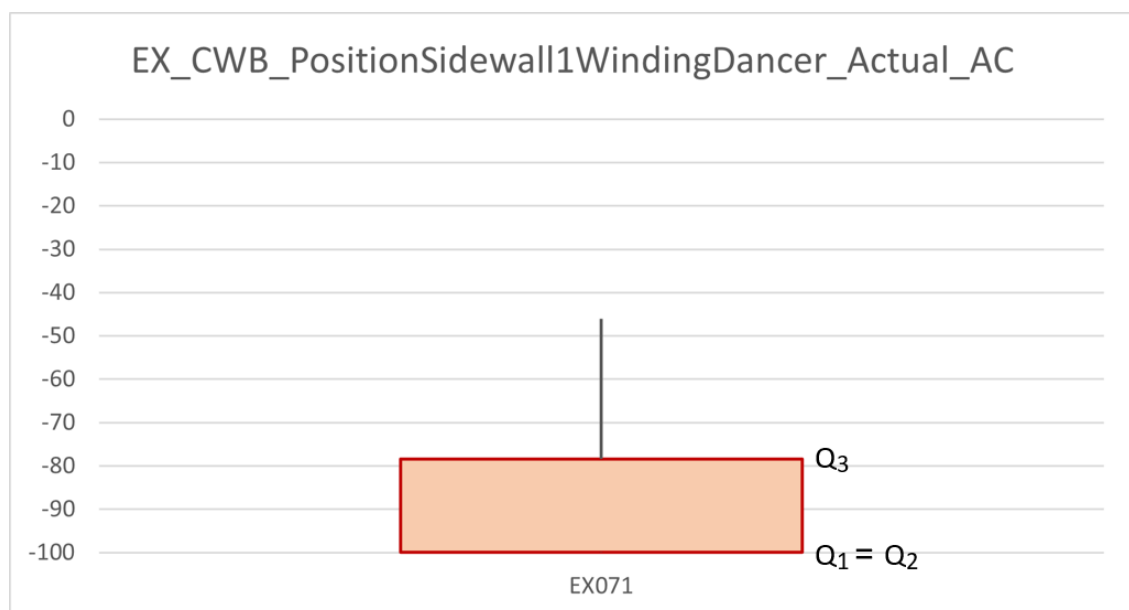


Fig. 5.9 Box diagram of the EX071 station for parameter a)

Parameter a) represents the position of the material. Based on the results, it can be concluded that it is determined by negative values. Up to the median level, the values from the minimum take on almost the same number. It is worth noting the zero number of null values.

#### b) EX\_CWB\_SpeedConveyor1\_Actual\_AC

Table 5.12 Analysis of the EX071 station for parameter b)

summary	EX071
Count	80,258,076
mean	4.98
stddev	6.05
Min	0
25%	0
50%	0
75%	10.41
max	37.45
nulls	0
skewness	right

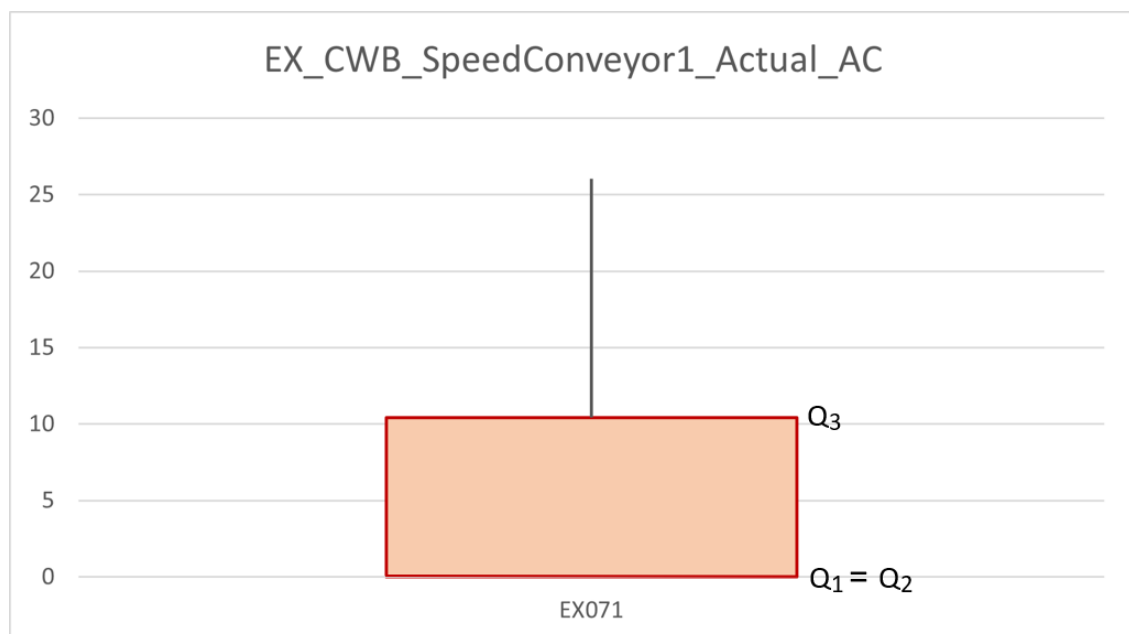


Fig. 5.10 Box diagram of the EX071 station for parameter b)

Parameter b) tells the speed of the conveyor. From Table 5.12, we can observe that at least half of the values reach level 0. Also for this case, the station has all the

values of the parameter acquired – they do not contain null.

### c) EX\_DC\_ConductivityWaterInlet\_Actual\_AC

Table 5.13 Analysis of the EX071 station for parameter c)

summary	EX071
Count	80,258,076
mean	890.9
stddev	70.2
Min	0
25%	839.1
50%	873.8
75%	954.9
max	1,365.7
nulls	0
skewness	right

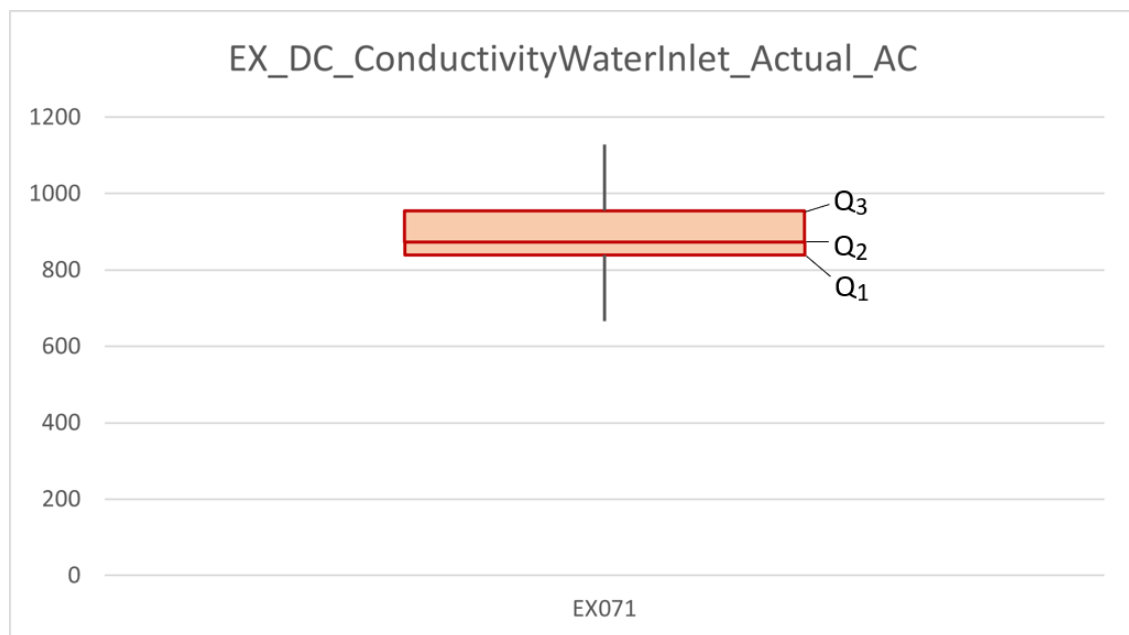


Fig. 5.11 Box diagram of the EX071 station for parameter c)

In the case of parameter c) we again have a zero number of null values. By comparing the distances between the quartiles from the graphical representation, as well as by comparing the values of the mean and median, we can conclude that this is a distribution of data with a skew to the right.

### d) EX\_DC\_pHValueWaterInlet\_Actual\_AC

Table 5.14 Analysis of the EX071 station for parameter d)

summary	EX071
Count	80,258,054
mean	6.7
stddev	0.6
Min	-2.0
25%	6.4
50%	6.7
75%	7.0
max	14.2
nulls	22
skewness	left

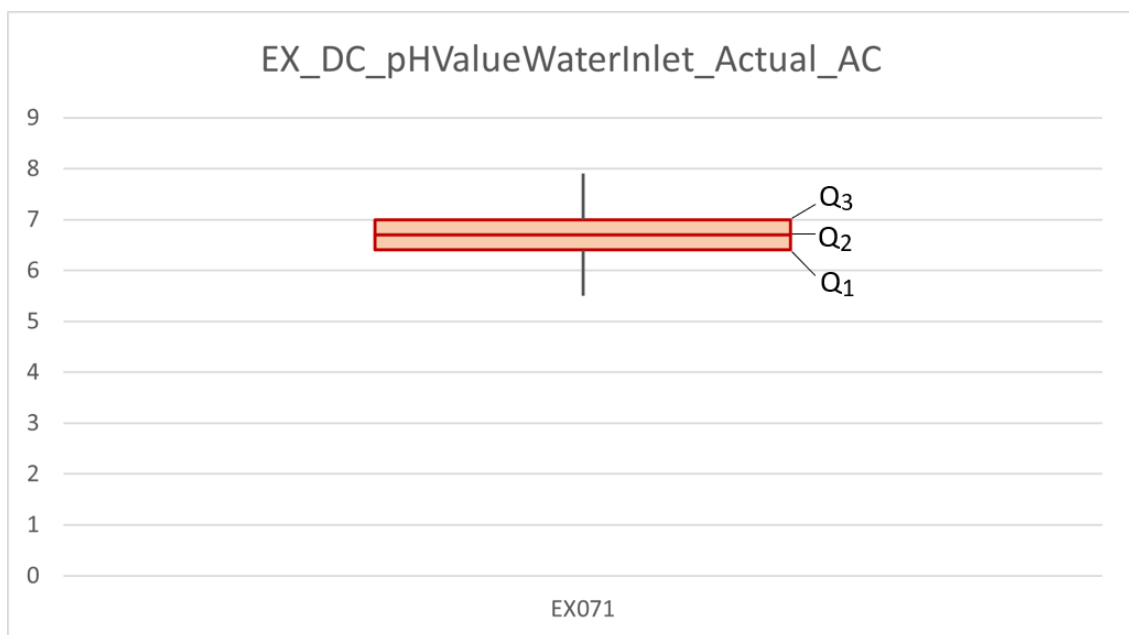


Fig. 5.12 Box diagram of the EX071 station for parameter d)

Parameter (d) records the pH values of the liquid used in the manufacturing process. Based on the values of the quartiles, we can see the density of points in the interquartile range ranging from level 6.4 to level 7. Thanks to these values, the relevant parameter achieves a low standard deviation level.

### e) EX\_EX2\_SpeedScrew\_Actual\_AC

Table 5.15 Analysis of the EX071 station for parameter e)

summary	EX071
Count	80,254,476
mean	6.5
stddev	10.0
Min	0
25%	0
50%	0
75%	11.3
max	48.0
nulls	3,600
skewness	right

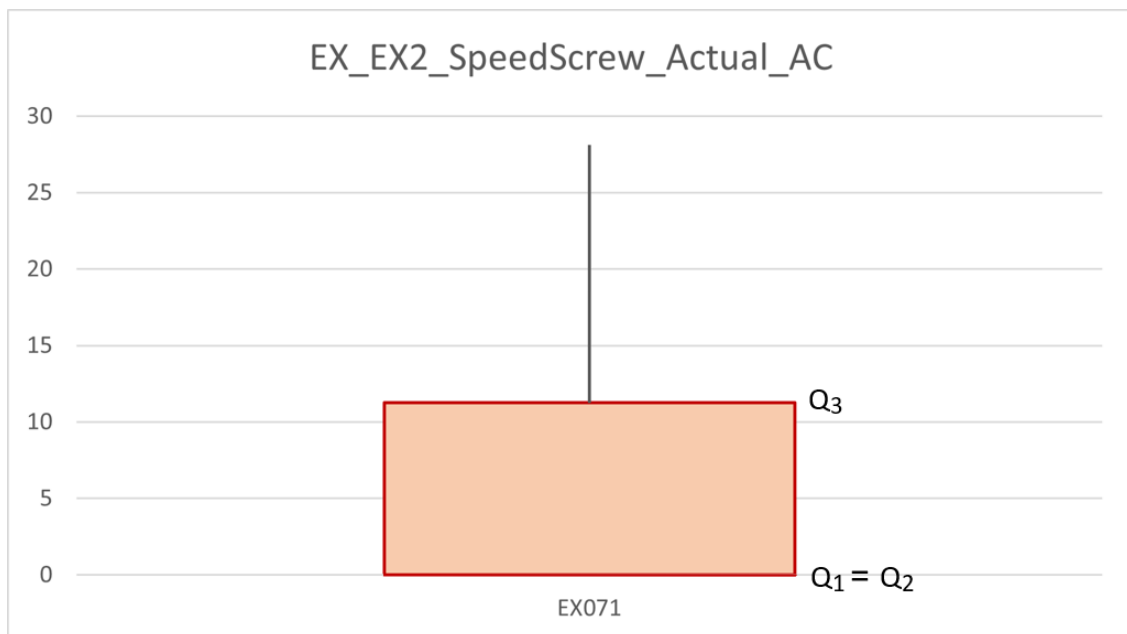


Fig. 5.13 Box diagram of the EX071 station for parameter e)

As we can see from Table 5.15, parameter e) reaches up to the median value of 0. The largest difference between the values can be observed between the third quartile and the maximum. As the numbers increase, the standard deviation level after rounding is 9.9966.



**f) EX\_CWB\_PressureSidewall1Winding\_Setpoint\_AC**

Table 5.16 Analysis of the EX071 station for parameter f)

summary	EX071
Count	80,256,456
mean	23.3
stddev	7.0
Min	-20
25%	25
50%	25
75%	25
max	2,540
nulls	1,620
skewness	left

Parameter f) represents the pressure in the production process. Based on the minimum value, it can be concluded that this is probably an error, as it is negative. The maximum value also looks suspicious. It is significantly higher than the values of quartiles. From the values of quartiles, we can state that the most frequently occurring level of measured points is 25.

## g) EX\_DC\_PositionDancer11\_Actual\_AC

Table 5.17 Analysis of the EX071 station for parameter g)

summary	EX071
Count	80,258,076
mean	-13.6
stddev	15.2
Min	-100
25%	-22.1
50%	-21.3
75%	-0.6
max	100
nulls	0
skewness	right

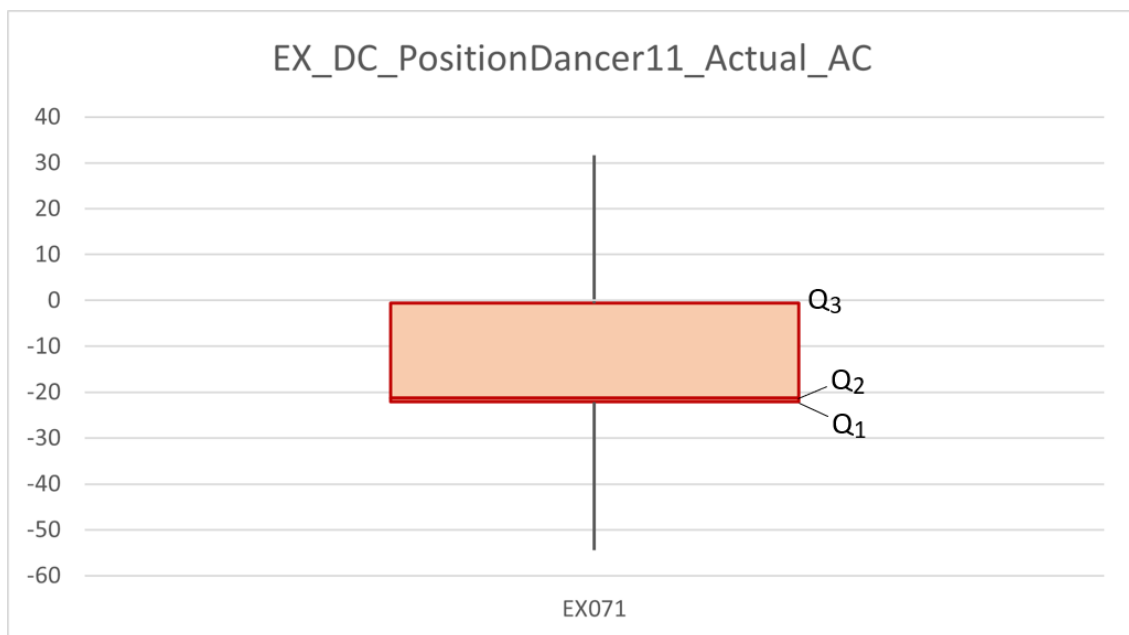


Fig. 5.14 Box diagram of the EX071 station for parameter g)

Parameter g) represents the position of the material. Based on the negative values up to the third quartile, it can be concluded that the position is mostly determined by them.

## h) EX\_DC\_PressureDancer1\_Actual\_AC

Table 5.18 Analysis of the EX071 station for parameter h)

summary	EX071
Count	80,258,054
mean	5.4
stddev	5.4
Min	0
25%	0
50%	5
75%	5
max	81
nulls	22
skewness	right

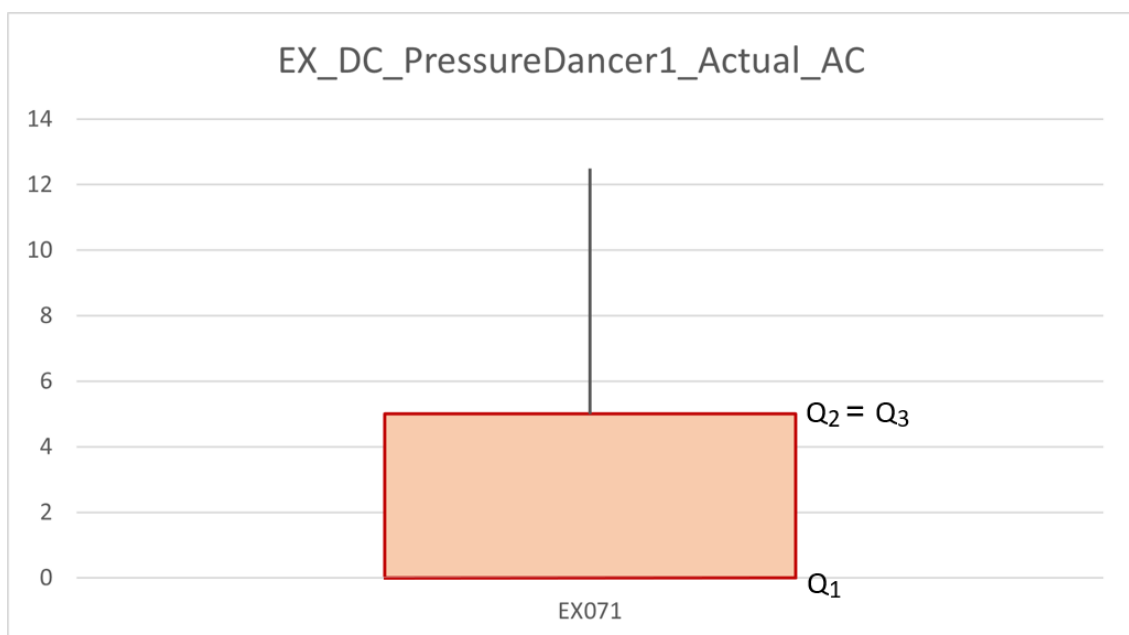


Fig. 5.15 Box diagram of the EX071 station for parameter h)

Parameter h) represents the pressure in the production process. For the first time, the quartile takes on values of 0. Subsequently, it has a level of 5 from the median to the third quartile. The numbers shown represent the most common values. The distribution of data is skewed to the right.

**i) EX\_DC\_TemperatureWaterDipCooling2\_Actual\_AC**

Table 5.19 Analysis of the EX071 station for parameter i)

summary	EX071
Count	80,256,276
mean	18.5
stddev	2.0
Min	0
25%	17.3
50%	17.9
75%	18.9
max	327.7
nulls	1,800
skewness	right

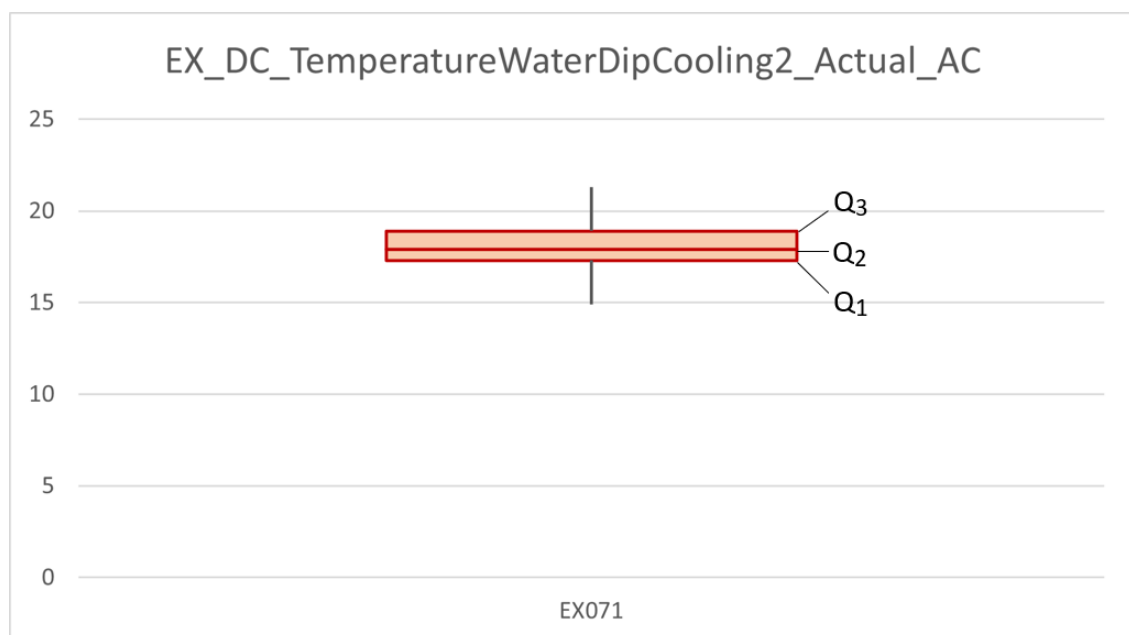


Fig. 5.16 Box diagram of the EX071 station for parameter i)

Parameter (i) represents the temperature in the production process. The maximum value looks suspicious. It is much higher compared to the third quartile. It is the distribution of data with a skew to the right.

### j) EX\_EX1\_PressureMassPin10\_Actual\_AC

Table 5.20 Analysis of the EX071 station for parameter j)

summary	EX071
Count	80,256,276
mean	25.8
stddev	59.8
Min	-414.8
25%	5.3
50%	5.6
75%	6.1
max	394.2
nulls	1,800
skewness	right

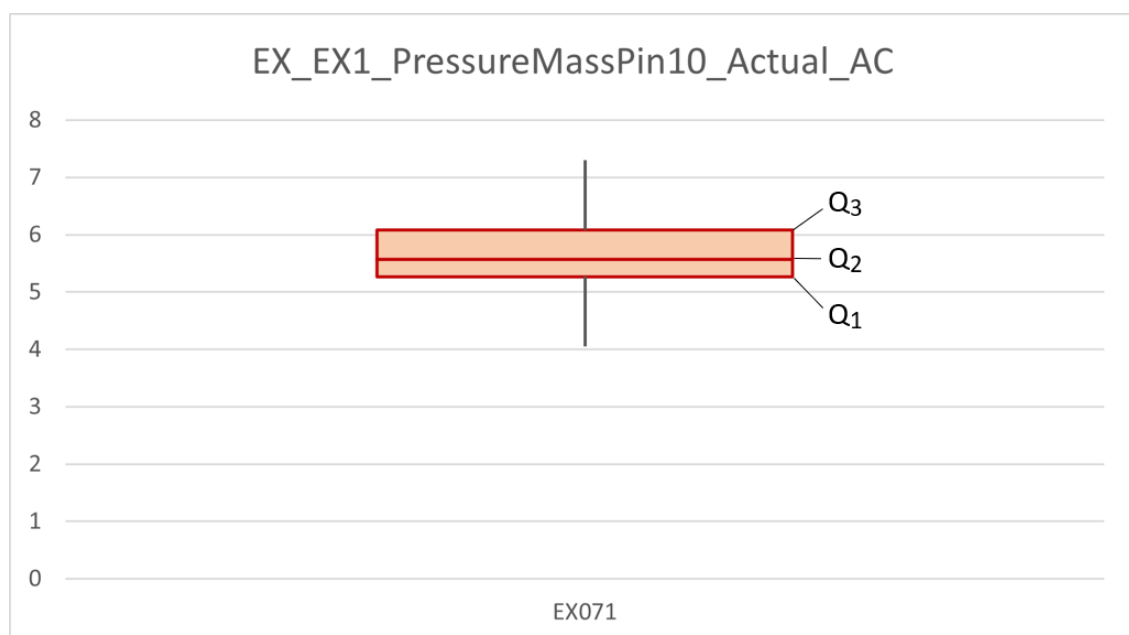


Fig. 5.17 Box diagram of the EX071 station for parameter j)

Parameter (j) expresses the pressure in the production process. When we look at the values of the quartiles, we can see that the numbers range from the level of 5.27 to the level of 6.08. The minimum and maximum values are significantly different, and the minimum is also negative. Most likely, these are mistakes.

### 5.3 Tire Molding

From the production process of tire molding, we decided to analyze the data of the TBPH1 and TBPH2 stations. Since most of the examined parameters contained null or 0 values, we analyzed only 6 parameters:

#### a) TB2\_CPL\_TensionH1\_Actual\_AC

Table 5.21 Analysis of TBPH1, TBPH2 stations for parameter a)

summary	TBPH1	TBPH2
Count	81,260,809	74,312,569
mean	2.3	5.5
stddev	12.2	26.7
Min	-227.2	-200.5
25%	-1.2	-0.6
50%	-0.1	0.3
75%	0.5	1.2
max	267.0	251
nulls	10,800	1,800
skewness	right	right

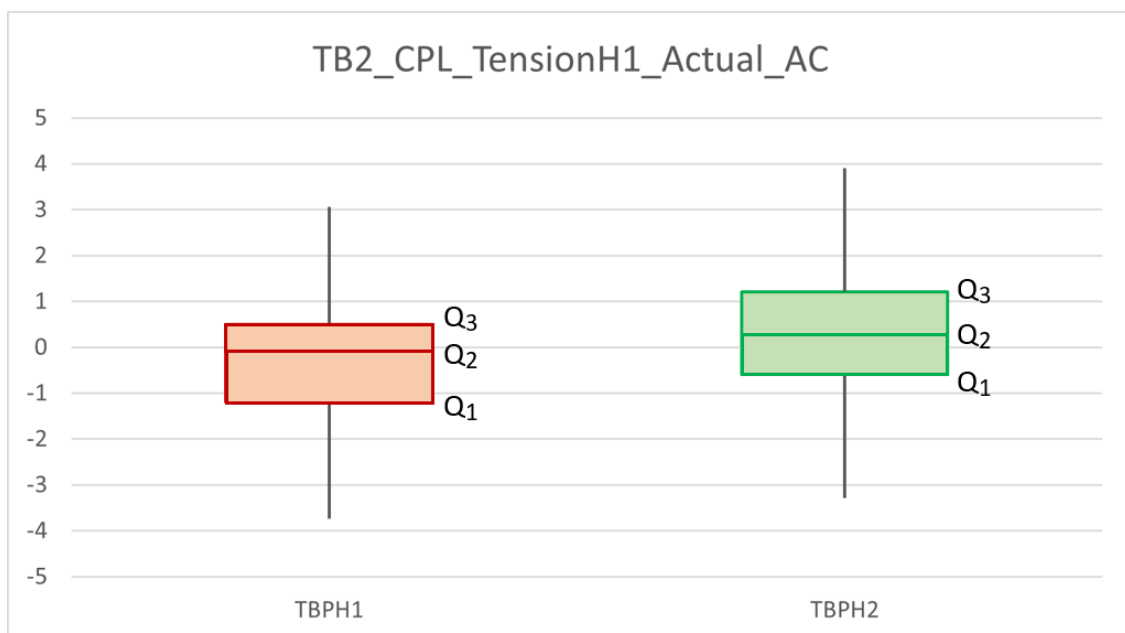


Fig. 5.18 Box diagram of stations TBPH1, TBPH2 for parameter a)

When we look at the results achieved for parameter a), we can see that the quartiles are around 0. On the contrary, the minimum and maximum are very far from the quartiles. The TBPH2 station acquires more than twice the standard deviation value compared to the TBPH1 station. It can be concluded that the TBPH2 station has more remote points, or their values are further away from the average, as is the case with the TBPH1 station.

### b) TB2\_CPL\_TensionH2\_Actual\_AC

Table 5.22 Analysis of TBPH1, TBPH2 stations for parameter b)

summary	TBPH1	TBPH2
Count	81,260,809	74,312,569
mean	0.8	4.4
stddev	11.4	13.3
Min	-171.5	-88.4
25%	-2.4	-1.0
50%	-0.4	0.2
75%	0.6	1.8
max	336.2	282.7
nulls	10,800	1,800
skewness	right	right

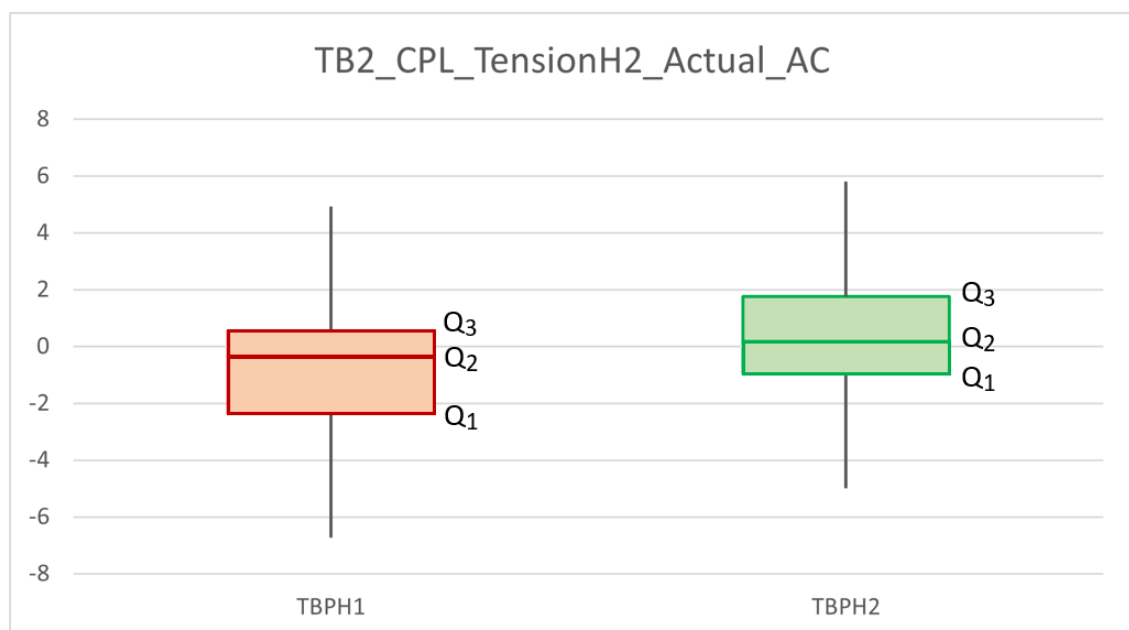


Fig. 5.19 Box diagram of stations TBPH1, TBPH2 for parameter b)

As was the case with parameter a), the values of the quartiles of both stations are kept around the value of 0. When we look at Figure 5.19, we can see that in the TBPH2 station, the quartiles acquire slightly higher values compared to the TBPH1 station. In both cases, it is a distribution of data with skewness to the right.

### c) TB2\_CPL\_TensionH3\_Actual\_AC

Table 5.23 Analysis of TBPH1, TBPH2 stations for parameter c)

summary	TBPH1	TBPH2
Count	80,500,221	74,312,569
mean	-11.9	26.7
stddev	27.7	56.7
Min	-183.3	-50.5
25%	-5.93	0
50%	0	0
75%	0	19.0
max	75.6	379.7
nulls	771,388	1,800
skewness	left	right

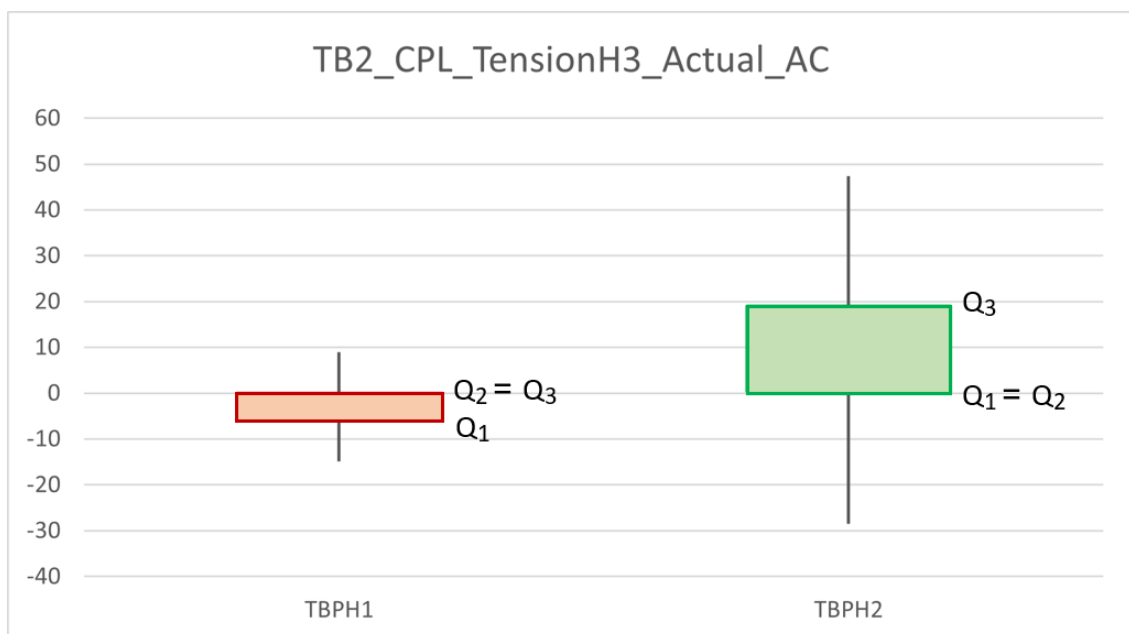


Fig. 5.20 Box diagram of stations TBPH1, TBPH2 for parameter c)

The difference between TBPH1 and TBPH2 stations for parameter c) can best be seen in Figure 5.20. For the TBPH1 station, the first quartile is negative and the second



together with the third quartiles take a value of 0. In the TBPH2 station, the first quartile together with the second quartile takes on a value of 0, while the third quartile has a level of 18.97.

#### d) TB2\_CPL\_TensionH4\_Actual\_AC

Table 5.24 Analysis of TBPH1, TBPH2 stations for parameter d)

summary	TBPH1	TBPH2
Count	80,500,221	74,312,569
mean	27.8	-2.3
stddev	66.9	75.9
Min	-33.0	-265.5
25%	0	0
50%	0	0
75%	11.6	0
max	380.1	265.5
nulls	771,388	1,800
skewness	right	left

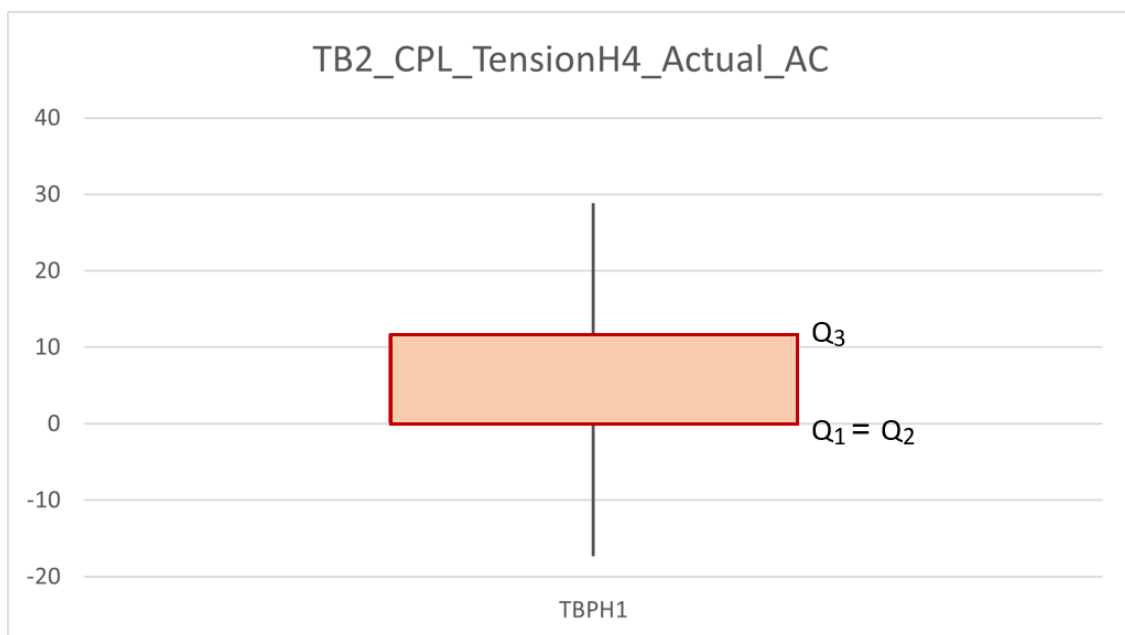


Fig. 5.21 Box diagram of TBPH1 stations for parameter d)

For parameter d) we can notice that we have only a graph to the TBPH1 station shown in the picture. This is because the TBPH2 station has a value of 0 for all quartiles. Thus, it has a zero magnitude of the interquartile distance from which it derives

Drawing a box diagram. Zero is the most common value in this case. In the case of the TBPH1 station, we have a third quartile that is non-zero. It acquires a value of 11.55. It is also possible to notice from the last row of Table 5.24 the differences in the direction of inclination between the stations. The TBPH1 station has the data distributed with a skew to the right, while the TBPH2 station has a skew to the left.

### e) TB2\_BR\_WidthStrip\_Setpoint\_AC

Table 5.25 Analysis of TBPH1, TBPH2 stations for parameter e)

summary	TBPH1	TBPH2
Count	81,251,339	74,314,369
mean	251.8	262.0
stddev	29.5	29.9
Min	0	0
25%	226	241
50%	246	256
75%	280	286
max	330	331
nulls	20,270	0
skewness	right	right

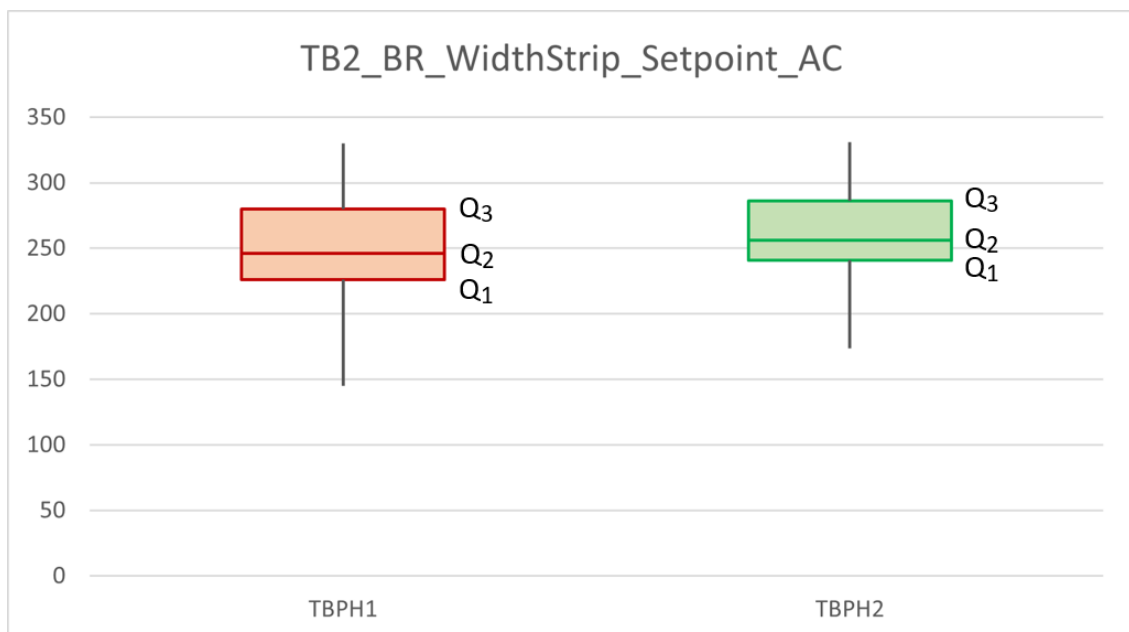


Fig. 5.22 Box diagram of stations TBPH1, TBPH2 for parameter e)

For parameter e), the similarity between TBPH1 and TBPH2 stations can be observed. They acquire the same minimum value and approximately the same maximum value. Their quartiles do not differ significantly either, with the difference that the TBPH1 station has lower levels.

#### f) TB2\_BL\_WidthStrip\_Setpoint\_AC

Table 5.26 Analysis of TBPH1, TBPH2 stations for parameter f)

summary	TBPH1	TBPH2
Count	81,271,609	74,314,369
mean	239.9	250.0
stddev	29.5	29.9
Min	0	0
25%	214	229
50%	234	244
75%	269	274
max	319	319
nulls	0	0
skewness	right	right

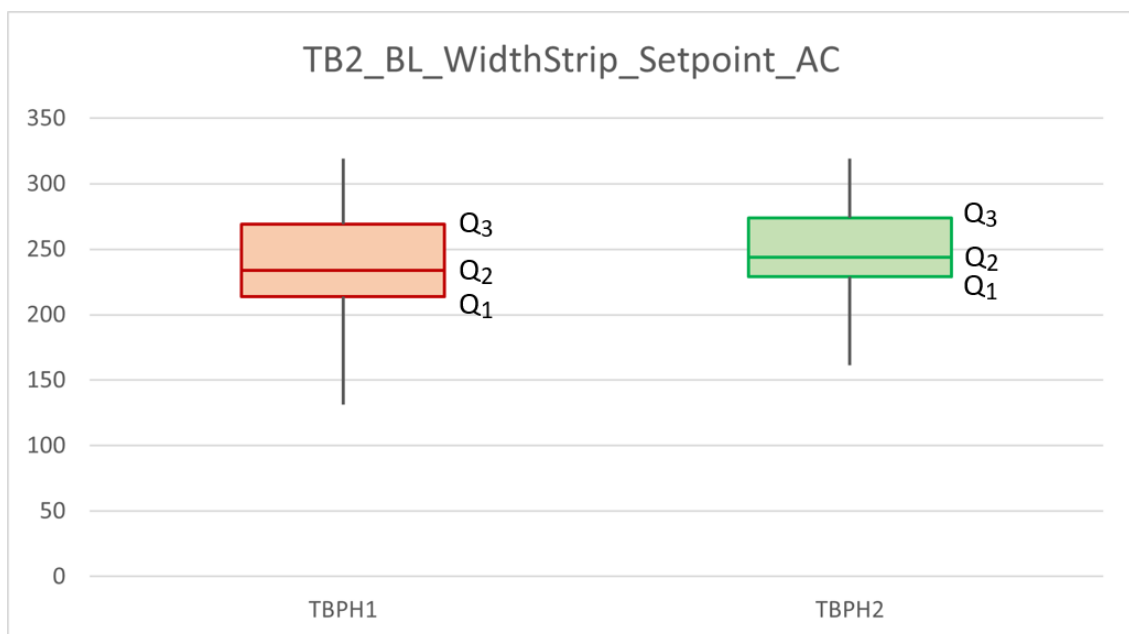


Fig. 5.23 Box diagram of stations TBPH1, TBPH2 for parameter f)

For parameter f) it is again possible to observe the similarity between the TBPH1 and TBPH2 stations in the achieved results. They acquire the same minimum and maximum, and similarly as in the case of parameter e), the levels of quartiles do not differ

---

significantly. Again the TBPH1 station has their values slightly lower compared to the TBPH2 station. This can be visually observed in Figure 5.23.

## CONCLUSION

In our diploma thesis, we introduced the area of big data. Their processing plays an important role in various industries. Based on it, it is possible to improve the quality of the industry, either by detecting shortcomings or predicting values. We dealt with the possibilities of processing big data. Programming languages containing various functions for working with data, as well as environments for distributed data processing (so-called *frameworks*), serve this purpose. In our work, we have introduced the possibilities and features of *the Python* and *Scala* programming languages. Both have their advantages that can be applied in the field. We have described *the Apache Spark* and *Ray environments*.

One of the goals of the thesis was to create an experimental workplace in the environment of the Continental data lake, using the services of AWS. We got acquainted with the possibilities that this environment offers. It provided us with the necessary storage for the data of the selected Continental production plant, which we processed and analyzed at a later stage of the project. We also used computing capacities in the form of instances. First, we got acquainted with the possibilities offered by the instance itself, later we created a cluster using them.

After the creation of the experimental workplace, we were able to continue analyzing the data of the manufacturing company. For this purpose, we used the created EMR *cluster*, into which we installed *Apache Spark* with five working nodes. For data processing, we used scripts written by the thesis consultant, Erick Solano Mora, PhD, in the *Scala programming language*. The data we had at our disposal was from three production areas – from the press shop, from extrusion and from confectionery, collected between 2020 and 2023. We used one of the scripts to list parameters for specific industries, the other script for the data processing itself. Data from 14 stations, each described by 58 parameters, where each parameter consisted of more than 80 million values, were recorded in the data lake. The extrusion area had data from a single station described by 229 parameters in the data lake, where each of them again consisted of more than 80 million values. Similarly, the ready-to-wear area consisted of four stations described by 51 parameters, each of which again had approximately 80 million values. As the total number of values in the data lake was very extensive, the aim of the thesis was not to process all of the parameters and stations. The work consultant gave us which of the stations we should focus on and determined that we should choose 10 parameters for analysis for each. We have also presented the results of all monitored parameters in the work.

The main contribution of the diploma thesis are the results of the processing of selected data of the Continental production plant in Korbach. Thanks to working with *Apache Spark*, we were able to process such a huge amount of data in a matter of seconds, which points to its power and the necessity of using this type of environment for working with big data. Based on the analysis of the mentioned data, it was possible to track whether the given parameters contained real data or whether there were also erroneous data. In the case of CU\_CU\_TemperatureADTT\_Actual\_AC temperature or pressures CU\_CU\_PressureInBladder\_Actual\_AC, EX\_CWB\_PressureSidewall1Winding\_Setpoint\_AC, EX\_EX1\_PressureMassPin10\_Actual\_AC the minimum negative values appeared, from which we concluded that it was most likely some kind of measurement error. According to the values of quartiles and minimums, together with maximums, it was possible to build box diagrams. Based on the above values in combination with a graphical representation, we were able to find out whether the variable contained outliers. In addition to the above conclusions, we compared each parameter between the monitored stations of the manufacturing industry. We were able to compare individual stations and find out their similarity or differences between them. In some cases, it happened to us that the monitored parameter had one type of skewness at one station, while in others it was the opposite (CU\_MO\_DurationShaping\_Actual\_AC, CU\_LOA\_DegreeInclinationSensorLoader\_Actual\_AC, CU\_MO\_ForceClosing\_Actual\_AC, TB2\_CPL\_TensionH3\_Actual\_AC, TB2\_CPL\_TensionH4\_Actual\_AC). In other cases, we were able to find approximately the same size of median and mean, which resulted in a normal distribution of data for a given parameter of the respective station (CU\_MO\_DurationShaping\_Actual\_AC: stations CU306, CU310; CU\_LOA\_DegreeInclinationSensorLoader\_Actual\_AC: stations CU306, CU307).

All the results achieved will be further used by Continental.

## List of bibliography

- [1] **Python.** What is Python? Executive Summary. *Python*. [Online]. [Retrieval date: 3.11.2023]. Available on the Internet: <<https://www.python.org/doc/essays/blurb/>>.
- [2] **Baeldung.** Statically Typed vs Dynamically Typed Languages. *Baeldung*. [Online] 17.5.2023. [Retrieval date: 3.11.2023]. Available on the Internet: <<https://www.baeldung.com/cs/statically-vs-dynamically-typed-languages>>.
- [3] **GeeksforGeeks.** Segmentation Fault in C/C++. *GeeksforGeeks*. [Online] 7.5.2023. [Retrieval date: 3.11.2023]. Available on the Internet: <<https://www.geeksforgeeks.org/segmentation-fault-c-cpp/>>.
- [4] **toppr.** Interpreter. *toppr*. [Online]. [Retrieval date: 3.11.2023]. Available on the Internet: <<https://www.toppr.com/guides/computer-science/computer-fundamentals/system-software/interpreter/>>.
- [5] **Malhotra, Aditi.** 5 Reasons Why You Should Choose Python for Big Data. *Whizlabs*. [Online]. [Retrieval date: 3.11.2023]. Available on the Internet: <<https://www.whizlabs.com/blog/python-and-big-data/>>.
- [6] **Tarud, Jonathan.** Open-Source Programming Languages Explained. *Koombea*. [Online] 1.2.2023. [Retrieval date: 3.11.2023]. Available on the Internet: <<https://www.koombea.com/blog/open-source-programming-languages-explained/>>.
- [7] **Yerukala, Madhuri.** Why is Python a perfect choice for Big Data? *Geospatial World*. [Online] 26.11.2020. [Retrieval date: 3.11.2023]. Available on the Internet: <<https://www.geospatialworld.net/blogs/why-is-python-a-perfect-choice-for-big-data/>>.
- [8] **GeeksforGeeks.** Pandas Introduction. *GeeksforGeeks*. [Online] 23.4.2024. [Retrieval date: 6.11.2023]. Available on the Internet: <<https://www.geeksforgeeks.org/introduction-to-pandas-in-python/>>.
- [9] **Ali Awan, Abid.** What is Labeled Data? *datacamp*. [Online] July 2023. [Retrieval date: 6.11.2023]. Available on the Internet: <<https://www.datacamp.com/blog/what-is-labeled-data>>.
- [10] **W3Schools.** Pandas Introduction. *W3Schools*. [Online]. [Retrieval date: 6.11.2023]. Available on the Internet: <[https://www.w3schools.com/python/pandas/pandas\\_intro.asp](https://www.w3schools.com/python/pandas/pandas_intro.asp)>.

- 
- [11] **docs.tibco**. Pivoting Data. *docs.tibco*. [Online]. [Retrieval date: 6.11.2023]. Available online: <[https://docs.tibco.com/pub/spotfire/6.5.2/doc/html/data/data\\_pivoting\\_data.htm](https://docs.tibco.com/pub/spotfire/6.5.2/doc/html/data/data_pivoting_data.htm)>.
- [12] **NumPy**. NumPy: the absolute basics for beginners. *NumPy*. [Online]. [Retrieval date: 6.11.2023]. Available on the Internet: <[https://numpy.org/doc/stable/user/absolute\\_beginners.html](https://numpy.org/doc/stable/user/absolute_beginners.html)>.
- [13] **GeeksforGeeks**. Introduction to NumPy. *GeeksforGeeks*. [Online] 14.8.2023. [Retrieval date: 6.11.2023]. Available on the Internet: <<https://www.geeksforgeeks.org/introduction-to-numpy/?ref=lbp>>.
- [14] **scikit-learn**. *scikit-learn: Machine Learning in Python*. [Online]. [Date of receipt: 7.11.2023]. Available online: <<https://scikit-learn.org/stable/>>.
- [15] **scikit-learn**. Getting Started. *scikit-learn: Machine Learning in Python*. [Online]. [Date of receipt: 7.11.2023]. Available on the Internet: <[https://scikit-learn.org/stable/getting\\_started.html](https://scikit-learn.org/stable/getting_started.html)>.
- [16] **scikit-learn**. Classifier comparison. *scikit-learn: Machine Learning in Python*. [Online]. [Date of receipt: 7.11.2023]. Available online: <[https://scikit-learn.org/stable/auto\\_examples/classification/plot\\_classifier\\_comparison.html](https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html)>.
- [17] **Seldon**. Machine Learning Regression Explained. *Seldon*. [Online] 29.10.2021. [Date of receipt: 7.11.2023]. Available on the Internet: <<https://www.seldon.io/machine-learning-regression-explained>>.
- [18] **scikit-learn**. Decision Tree Regression with AdaBoost. *scikit-learn: Machine Learning in Python*. [Online]. [Date of receipt: 7.11.2023]. Available on the Internet: <[https://scikitlearn.org/stable/auto\\_examples/ensemble/plot\\_adaboost\\_regression.html](https://scikitlearn.org/stable/auto_examples/ensemble/plot_adaboost_regression.html)>.
- [19] **scikit-learn**. Comparing different clustering algorithms on toy datasets. *scikit-learn: Machine Learning in Python*. [Online]. [Date of receipt: 7.11.2023]. Available on the Internet: <[https://scikitlearn.org/stable/auto\\_examples/cluster/plot\\_cluster\\_comparison.html#sphx-glr-auto-examples-cluster-plot-cluster-comparison-py](https://scikitlearn.org/stable/auto_examples/cluster/plot_cluster_comparison.html#sphx-glr-auto-examples-cluster-plot-cluster-comparison-py)>.
- [20] **GeeksforGeeks**. Introduction to Dimensionality Reduction. *GeeksforGeeks*. [Online] 6.5.2023. [Date of receipt: 7.11.2023]. Available on the Internet: <<https://www.geeksforgeeks.org/dimensionality-reduction/>>



- [21] **scikit-learn**. Demonstration of multi-metric evaluation on cross\_val\_score and GridSearchCV. *scikit-learn: Machine Learning in Python*. [Online]. [Date of receipt: 7.11.2023]. Available on the Internet: <[https://scikitlearn.org/stable/auto\\_examples/model\\_selection/plot\\_multi\\_metric\\_evaluation.html](https://scikitlearn.org/stable/auto_examples/model_selection/plot_multi_metric_evaluation.html)>.
- [22] **SciPy**. *SciPy*. [Online]. [Retrieval date: 8.11.2023]. Available on the Internet: <<https://scipy.org/>>.
- [23] **mlpy**. mlpy – Machine Learning Python. *MLPY: Machine Learning PY*. [Online]. [Retrieval date: 8.11.2023]. Available on the Internet: <<https://mlpy.sourceforge.net/>>.
- [24] **matplotlib**. Matplotlib: Visualization with Python. *matplotlib*. [Online]. [Retrieval date: 8.11.2023]. Available online: <<https://matplotlib.org/>>.
- [25] **GeeksforGeeks**. Introduction to Matplotlib. *GeeksforGeeks*. [Online] 12.3.2023. [Retrieval date: 8.11.2023]. Available on the Internet: <<https://www.geeksforgeeks.org/python-introduction-matplotlib/>>.
- [26] **TensorFlow**. Introduction to TensorFlow. *TensorFlow*. [Online]. [Retrieval date: 8.11.2023]. Available on the Internet: <<https://www.tensorflow.org/learn>>.
- [27] **GeeksforGeeks**. Introduction to TensorFlow. *GeeksforGeeks*. [Online] 18.6.2023. [Retrieval date: 8.11.2023]. Available on the Internet: <<https://www.geeksforgeeks.org/introduction-to-tensorflow/>>.
- [28] **GeeksforGeeks**. Theano and Python. *GeeksforGeeks*. [Online] 16.8.2023. [Retrieval date: 8.11.2023]. Available on the Internet: <<https://www.geeksforgeeks.org/theano-in-python/>>.
- [29] **GeeksforGeeks**. NetworkX: Python software package for study of complex networks. *GeeksforGeeks*. [Online] 11.7.2022. [Retrieval date: 8.11.2023]. Available on the Internet: <<https://www.geeksforgeeks.org/networkx-python-software-package-study-complex-networks/>>.
- [30] **GeeksforGeeks**. Python: Getting started with SymPy module. *GeeksforGeeks*. [Online] 21.4.2022. [Retrieval date: 8.11.2023]. Available on the Internet: <<https://www.geeksforgeeks.org/python-getting-started-with-sympy-module/>>.
- [31] **GeeksforGeeks**. Dask and Python. *GeeksforGeeks*. [Online] 16.11.2022. [Retrieval date: 8.11.2023]. Available on the Internet: <<https://www.geeksforgeeks.org/introduction-to-dask-in-python/>>.

- [32] **GeeksforGeeks**. Introduction to Scala. *GeeksforGeeks*. [Online] 24.9.2021. [Retrieval date: 15.11.2023]. Available on the Internet: <<https://www.geeksforgeeks.org/introduction-to-scala/>>.
- [33] **Jimenez, Andrés**. What are Primitive Data Types? *Devlane*. [Online] July 2023. [Retrieval date: 15.11.2023]. Available on the Internet: <<https://www.devlane.com/blog/what-are-primitive-data-types>>.
- [34] **docs.scala**. Tour of Scala: Type Inference. *docs.scala*. [Online]. [Retrieval date: 15.11.2023]. Available on the Internet: <<https://docs.scala-lang.org/tour/type-inference.html>>.
- [35] **scala-lang**. *scala-lang*. [Online]. [Retrieval date: 15.11.2023]. Available online: <<https://www.scala-lang.org/>>.
- [36] **DataScientest**. Comparing Scala and Python: Choosing the Right Language for Your Projects. *DataScientest*. [Online] 1.1.2023. [Date of receipt: 22.11.2023]. Available on the Internet: <<https://datascientest.com/en/comparing-scala-and-python-choosing-the-right-language-for-your-projects> >.
- [37] **Narayanan, Raji**. Python vs. Scala: A Deep Dive Comparison. *StreamSets: A Software AG Company*. [Online] 11.3.2022. [Date of receipt: 22.11.2023]. Available online: <<https://streamsets.com/blog/python-vs-scala-comparison/#Python-vs.-Scala-Comparison>>.
- [38] **nglogic**. Scala vs. Python: What's Better for Big Data? *nglogic*. [Online]. [Date of receipt: 22.11.2023]. Available on the Internet: <<https://nglogic.com/scala-vs-python/>>.
- [39] **Apache Hadoop**. *Apache Hadoop*. [Online]. [Date of receipt: 22.11.2023]. Available online: <<https://hadoop.apache.org/>>.
- [40] **Techreviewer**. The Most Popular Big Data Frameworks in 2023. *Techreviewer*. [Online] 29.3.2023. [Retrieval date: 29.11.2023]. Available on the Internet: <<https://techreviewer.co/blog/the-most-popular-big-data-frameworks>>.
- [41] **Davies, Blake**. Best Data Processing Frameworks That You Must Know. *upGrad: KnowledgeHut*. [Online] 12.10.2023. [Retrieval date: 29.11.2023]. Available on the Internet: <<https://www.knowledgehut.com/blog/big-data/5-best-data-processing-frameworks> >.
- [42] **Ray**. Scaling AI and Python Workloads Effortlessly with Ray. *Ray*. [Online] 2022. [Date of receipt: 5.1.2024]. Available on the Internet: <[https://assets.ctfassets.net/bguokct8bxgd/26Vuu2NjLVnWkX4TkaISmB/fbc74da45885ca8e5048583f8a7e9d25/Ray\\_OSS\\_Datasheet\\_-\\_Final.pdf](https://assets.ctfassets.net/bguokct8bxgd/26Vuu2NjLVnWkX4TkaISmB/fbc74da45885ca8e5048583f8a7e9d25/Ray_OSS_Datasheet_-_Final.pdf)>.

- [43] **PENIAK, P. – HOLEČKO, P. – BUBENÍKOVÁ, E.** tag. 2023. *Applications of Information Systems for Industry 4.0*. 1st edition. Žilina: EDIS, 2023. 300 pages. ISBN 978-80-554-1985-5.
- [44] **Apache Spark**. Apache Spark – A Unified engine for large-scale data analytics. *Apache Spark*. [Online]. [Retrieval date: 29.11.2023]. Available on the Internet: <<https://spark.apache.org/docs/latest/index.html>>.
- [45] **databricks**. Apache Spark. *databricks*. [Online]. [Retrieval date: 29.11.2023]. Available on the Internet: <<https://www.databricks.com/glossary/what-is-apache-spark>>.
- [46] **Rohan, Joseph**. What is Spark? *Chartio: Data Tutorials*. [Online]. [Retrieval date: 29.11.2023]. Available on the Internet: <<https://chartio.com/learn/data-analytics/what-is-spark/>>.
- [47] **databricks**. PySpark. *databricks*. [Online]. [Retrieval date: 4.12.2023]. Available online: <<https://www.databricks.com/glossary/pyspark>>.
- [48] **Apache Spark**. PySpark Documentation. *Apache Spark*. [Online]. [Retrieval date: 4.12.2023]. Available on the Internet: <<https://spark.apache.org/docs/3.3.1/api/python/index.html>>.
- [49] **Hewlett Packard Enterprise**. Workload. *Hewlett Packard Enterprise*. [Online]. [Retrieval date: 29.11.2023]. Available on the Internet: <<https://www.hpe.com/us/en/what-is/workload.html>>.
- [50] **aws**. What is Apache Spark? *aws*. [Online]. [Date of receipt: 2.12.2023]. Available online: <<https://aws.amazon.com/what-is/apache-spark/>>.
- [51] **Vaidya, Neha**. Apache Spark Architecture – Spark Cluster Architecture Explained. *edureka*. [Online] 26.10.2023. [Retrieval date: 13.12.2023]. Available online: <<https://www.edureka.co/blog/spark-architecture/>>.
- [52] **Thailappan, Dhanya**. Understand The Internal Working of Apache Spark. *Analytics Vidhya*. [Online] 26.8.2021. [Date of receipt: 1.1.2024]. Available online: <<https://www.analyticsvidhya.com/blog/2021/08/understand-the-internal-working-of-apache-spark/>>.
- [53] **Ray**. *Ray*. [Online]. [Date of receipt: 5.1.2024]. Available on the Internet: <<https://www.ray.io/>>.
- [54] **databricks**. Introduction to Data Lakes. *databricks*. [Online]. [Date of receipt: 25.3.2024]. Available online: <<https://www.databricks.com/discover/data-lakes>>.

- [55] **Harris, Jim**. What is a data lake and why does it matter? SAS. [Online]. [Date of receipt: 25.3.2024]. Available on the Internet: <[https://www.sas.com/en\\_us/insights/articles/data-management/what-is-a-data-lake-and-why-does-it-matter-.html](https://www.sas.com/en_us/insights/articles/data-management/what-is-a-data-lake-and-why-does-it-matter-.html)>.
- [56] **aws**. What is a Data Lake? aws. [Online]. [Date of receipt: 25.3.2024]. Available online: <<https://aws.amazon.com/what-is/data-lake/>>.
- [57] **BasuMallick, Chiradeep**. What Is Big Data? Definition, Types, Importance, and Best Practices. *spiceworks*. [Online] 8.8.2022. [Date of receipt: 25.3.2024]. Available online: <<https://www.spiceworks.com/tech/big-data/articles/what-is-big-data/>>.
- [58] **aws**. Cloud computing with AWS. aws. [Online]. [Date of receipt: 26.3.2024]. Available online: <[https://aws.amazon.com/what-is-aws/?nc1=f\\_cc](https://aws.amazon.com/what-is-aws/?nc1=f_cc)>.
- [59] **docs.aws**. What is Amazon S3? docs.aws. [Online]. [Retrieval date: 3.4.2024]. Available on the Internet: <<https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html>>.
- [60] **aws**. Amazon S3 Features. aws. [Online]. [Retrieval date: 3.4.2024]. Available online: <<https://aws.amazon.com/s3/features/>>.
- [61] **docs.aws**. Buckets overview. docs.aws. [Online]. [Retrieval date: 3.4.2024]. Available on the Internet: <<https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingBucket.html#bucket-config-options-intro>>.
- [62] **docs.aws**. Amazon S3 objects overview. docs.aws. [Online]. [Retrieval date: 3.4.2024]. Available on the Internet: <<https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingObjects.html>>.
- [63] **docs.aws**. What is the AWS Command Line Interface? docs.aws. [Online]. [Retrieval date: 3.4.2024]. Available on the Internet: <<https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html>>.
- [64] **aws**. AWS Command Line Interface. aws. [Online]. [Retrieval date: 3.4.2024]. Available online: <<https://aws.amazon.com/cli/>>.
- [65] **docs.aws**. Amazon Elastic Compute Cloud Documentation. docs.aws. [Online]. [Retrieval date: 28.3.2024]. Available on the Internet: <[https://docs.aws.amazon.com/ec2/?nc2=h\\_ql\\_doc\\_ec2](https://docs.aws.amazon.com/ec2/?nc2=h_ql_doc_ec2)>.
- [66] **docs.aws**. What is Amazon EC2? docs.aws. [Online]. [Retrieval date: 29.3.2024]. Available on the Internet: <<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>>.

- [67] **aws**. FAQs. *aws*. [Online]. [Retrieval date: 29.3.2024]. Available on the Internet: <<https://aws.amazon.com/ec2/faqs/#how-many-instances-ec2>>.
- [68] **docs.aws**. Instances and AMIs. *docs.aws*. [Online]. [Retrieval date: 29.3.2024]. Available on the Internet: <<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instances-and-amis.html> >.
- [69] **docs.aws**. Amazon EC2 instance types. *docs.aws*. [Online]. [Retrieval date: 29.3.2024]. Available on the Internet: <<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html>>.
- [70] **docs.aws**. Lifecycle instances. *docs.aws*. [Online]. [Retrieval date: 29.3.2024]. Available on the Internet: <<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-lifecycle.html> >.
- [71] **docs.aws**. What is Amazon EMR? *docs.aws*. [Online]. [Retrieval date: 3.4.2024]. Available on the Internet: <<https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-what-is-emr.html> >.
- [72] **docs.aws**. Overview of Amazon EMR. *docs.aws*. [Online]. [Retrieval date: 3.4.2024]. Available on the Internet: <<https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-overview.html> >.
- [73] **aws**. Amazon EMR features. *aws*. [Online]. [Retrieval date: 3.4.2024]. Available online: <<https://aws.amazon.com/emr/features/>>.
- [74] Continental Corporate Documentation (for the Data Lake)
- [75] **awscli**. Ls. *AWS CLI Command Reference*. [Online]. [Date of receipt: 16.4.2024]. Available on the Internet: <<https://awscli.amazonaws.com/v2/documentation/api/latest/reference/s3/ls.html>>.
- [76] **MobaXterm**. MobaXterm. *MobaXterm*. [Online]. [Date of receipt: 17.4.2024]. Available online: <<https://mobaxterm.mobatek.net/>>.
- [77] **Turney, Shaun**. Skewness: Definition, Examples & Formula. *Scribbr*. [Online]. [Retrieval date: 26.4.2024]. Available on the Internet: <<https://www.scribbr.com/statistics/skewness/>>.

# Annex

# List of attachments

Annex A | CD ..... II

## **Annex A | CD**

The CD contains an electronic version of the diploma thesis.